UNIVERSIDADE FEDERAL DO PARANÁ

ALINE KIESKOSKI

UM ESTUDO DO PROBLEMA DE *FLOW SHOP* PERMUTACIONAL. UMA PROPOSTA DE SOLUÇÃO ATRAVÉS DA METAHEURÍSTICA COLÔNIA DE FORMIGAS

CURITIBA

2016

ALINE KIESKOSKI

UM ESTUDO DO PROBLEMA DE *FLOW SHOP* PERMUTACIONAL. UMA PROPOSTA DE SOLUÇÃO ATRAVÉS DA METAHEURÍSTICA COLÔNIA DE FORMIGAS

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre em Métodos Numéricos em Engenharia, área de concentração em Programação Matemática, no Curso de Pós-Graduação em Métodos Numéricos em Engenharia, Setores de Tecnologia e Ciências Exatas, da Universidade Federal do Paraná.

Orientadora: Profa. Dra. Neida Maria Patias Volpi

CURITIBA

2016

Kieskoski, Aline
Um estudo do problema de *flow shop* permutacional, uma proposta de solução através da metaheurística colônia de formigas / Aline Kieskoski . – Curitiba, 2016.
109 f. : il., tabs.

Dissertação (mestrado) – Universidade Federal do Paraná, Setores de Tecnologia e de Ciências Exatas, Programa de Pós-Graduação em Métodos Numéricos em Engenharia.
Orientador: Neida Maria Patias Volpi
Bibliografia: p. 100-102

1. Programação (Matemática). 2. Programação heurística. 3. Otimização combinatória. I. Volpi, Neida Maria Patias. II. Título.

CDD 621.3191

Ministério da Educação
Universidade Federal do Paraná
Setor de Tecnologia / Setor de Ciências Exatas
Departamento de Construção Civil / Departamento de Matemática
Programa de Pós-Graduação em Métodos Numéricos em Engenharia – PPGMNE/UFPR.

PPGMNE

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Métodos Numéricos Em Engenharia da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de Aline Kieskoski, intitulada: "*Um Estudo do Problema de Flow Shop Permutacional. Uma Proposta de Solução Através da Metaheurística Colônia de Formigas*", após terem inquirido a aluna e realizado a avaliação do trabalho, são de parecer pela sua APROVAÇÃO.

Curitiba, 04 de outubro de 2016.

NEIDA MARIA PATIAS VOLPI (UFPR)
(Presidente da Banca Examinadora)

ARINEI CARLOS LINDBECK DA SILVA (UFPR)

LEANDRO MAGATÃO (UTFPR)

Curitiba, 04 de outubro de 2016

Agradecimentos

A Deus, que iluminou meu caminho, me deu saúde, força e coragem para concluir esta etapa de minha vida.

Aos meus pais, Silvestre e Janice, agradeço pelo dom da vida, pelo carinho e compreensão em todos os momentos de ausência. Obrigada pelo apoio e incentivo, vocês são meus alicerces!

Ao meu irmão André Kieskoski (*in memoriam*), que sempre incentivou meus estudos. Mesmo sem contato, nos momentos difíceis lembrei-me de quantas batalhas você venceu, isso me deu forças para ir até o fim. Sei que onde estiver, está orgulhoso da sua maninha.

Ao meu namorado Marcelo, agradeço por todo amor e carinho, pela compreensão nos momentos de ausência e por ouvir pacientemente todas as minhas angústias.

A minha orientadora, Profa. Dra. Neida Maria Patias Volpi, por sua disposição, atenção e pela dedicação na orientação deste trabalho.

Aos professores, Arinei e Leandro, pela revisão e considerações à este trabalho.

Aos professores e funcionários do Programa de Pós-Graduação em Métodos Numéricos em Engenharia.

Aos colegas e amigos, em especial: Carlos, Carolina, Dayane, Deidson, Nathália, Jocelaine, Joyce, Maria Carolina, Maria Cláudia, Ricardo e Sara, pela amizade e companheirismo.

A CAPES, pelo auxílio financeiro.

Resumo

Este trabalho estuda o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*. Descreve-se este problema em um modelo de programação linear. Este modelo é testado com o otimizador CPLEX, em um conjunto de problemas gerado aleatoriamente. Propõe-se um algoritmo, baseado na metaheurística colônia de formigas, com considerações a respeito dos tempos de *setup* nas regras de transição. Verifica-se a eficiência deste algoritmo, em relação à qualidade das soluções obtidas e, ao tempo de resolução fazendo-se comparações com os resultados obtidos com o modelo, no otimizador. A inicialização das trilhas de feromônios da metaheurística é feita a partir de soluções iniciais. As soluções iniciais são geradas com base em alguns métodos já propostos na literatura, com adicional de considerações a respeito dos tempos de *setup*. Testa-se a influência da consideração destes tempos de *setup* na qualidade destas soluções iniciais. Verificou-se que o modelo, resolvido com o otimizador CPLEX, utiliza um tempo computacional maior que o algoritmo colônia de formigas e, teve dificuldade em resolver instâncias de problemas maiores com a limitação de tempo em 3 horas. O algoritmo baseado na otimização colônia de formigas encontrou boas soluções em um tempo computacional pequeno.

Palavras-chave: *Flow shop* Permutacional. Sequência dependente dos tempos de *setup*. Metaheurística colônia de formigas.

Abstract

This paper studies the problem of permutation flowshop, with sequence- dependent setup times. Describes this problem in a linear programming model. This model is tested with CPLEX optimizer, in a set of randomly generated problems. Proposed an algorithm based on ant colony metaheuristic, with considerations about the setup times in the transitional rules. There is the efficiency of this algorithm on the quality of the solutions and the time resolution, by making comparisons with the results obtained with the model in the optimizer. In metaheuristic, the initialization of the pheromone trails is made from initial solutions. The initial solutions are generated based on some methods that have been proposed in the literature, with additional considerations regarding the setup times. Was tested the influence the consideration setup times in quality the initial solutions. It was found that the model solved with CPLEX optimizer uses more computational time that the ant colony algorithm, and they have difficulty in solving the major problems with the time limit in 3 hours. The algorithm based on ant colony optimization found good solutions in a short computational time.

Key-words: Permutation flowshop. Sequence-dependent setup times. Ant colony metaheuristic.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

O sistema produtivo depende de gerenciamento, executar esse gerenciamento envolve a necessidade de tomar decisões. O planejamento e controle da produção (PCP), é um apoio para a tomada de decisões no sistema produtivo que, deve ser planejado de forma a diminuir custos, otimizar o uso de recursos e, usar eficientemente a capacidade de produção, de modo a atender os clientes. Para que isso se concretize, decisões precisam ser tomadas, essas decisões envolvem por exemplo, escala de funcionários, administração de estoques, entrega de produtos, organização da sequência de tarefas.

A organização da sequência de tarefas faz parte do planejamento e controle da produção (PCP), mais especificamente, do nível operacional da produção, chamado de programação da produção. A pesquisa operacional desenvolve modelos e apresenta métodos que auxiliam o gerente de produção a obter melhores soluções de programação para cada problema específico, em um horizonte de tempo limitado.

Neste trabalho, será abordado um problema que envolve o sequenciamento de tarefas, o problema de *flow shop* permutacional. Neste ambiente de produção, consideram-se n tarefas, que devem ser processadas sequencialmente em m máquinas. Todas as tarefas são processadas em todas as máquinas e, a sequência de tarefas é a mesma em todas elas.

Dentro deste ambiente de produção, pode-se considerar alguns aspectos adicionais. Neste trabalho será considerada a situação em que existe um tempo de preparação de máquina (*setup*) entre as tarefas e, este tempo altera em função da sequência programada, esta condição é chamada de sequência dependente dos tempos de *setup*. Para avaliar a qualidade de um programa de produção utilizam-se medidas de desempenho, neste trabalho, a qualidade das soluções encontradas serão avaliadas de acordo com o valor do *makespan* (C_{max}), que é o instante de conclusão da última tarefa, na última máquina.

Na prática, o problema de *flow shop* permutacional é comum em indústrias que precisam de um sequenciamento eficiente de tarefas. Devido a sua importância, este problema vêm sendo estudado a décadas porém, os estudos considerando a sequência dependente dos tempos de *setup* são mais recentes (NEUFELD; GUPTA; BUSCHER, 2016). Esta aplicação é muito comum em indústrias gráficas ou de tintas que, utilizam um tempo muito grande de preparação das máquinas, dependendo da alteração da cor, tamanho ou material que está sendo produzido.

Neste trabalho, no Capítulo 2 faz-se um levantamento teórico sobre a programação da produção, as principais publicações acerca do problema de *flow shop*, bem como, os métodos de resolução deste problema, já propostos na literatura. No Capítulo 3, propõe-se um modelo (MFSP) que descreve o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*.

As primeiras resoluções para o problema de *flow shop* permutacional envolviam técnicas de enumeração porém, por se tratar de um problema NP-*hard* (LENSTRA; RINNOOY, 1977), o custo computacional envolvido é grande, limitando esta resolução somente a problemas de pequeno a médio porte. Para problemas maiores, pesquisadores propõem utilizar técnicas de otimização heurísticas ou metaheurísticas, que com um bom ajuste de parâmetros, produzem soluções de qualidade, próximas da melhor solução.

Neufeld, Gupta e Buscher (2016) fizeram uma revisão de literatura acerca das publicações dos problemas de *flow shop*. Dentre as técnicas já aplicadas para resolver o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, encontrou-se na literatura alguns trabalhos que resolveram este problema utilizando a metaheurística colônia de formigas como (GAJPAL; RAJENDRAN; ZIEGLER, 2006) e (MIRABI, 2011). Porém, a maior parte destes trabalhos não faz considerações explicitas dos tempos de *setup* nas regras de transição, que são as regras utilizadas para construir as sequências de tarefas. Acredita-se que estas considerações sejam importantes, portanto, no Capítulo 4 propõe-se um algoritmo (AFSP), baseado na metaheurística colônia de formigas, que faz considerações acerca dos tempos de *setup* nas regras de transição.

Para realizar testes com o modelo (MFSP) e, o algoritmo proposto (AFSP), gerou-se instâncias de problemas aleatoriamente. Os resultados dos experimentos computacionais estão expostos no Capítulo 5. As conclusões finais, acerca dos experimentos computacionais realizados, estão expostas no Capítulo 6. A seguir, apresenta-se os objetivos deste trabalho.

1.1 Objetivos

1.1.1 Objetivo geral

Fazer um estudo do problema de *flow shop* permutacional e, propôr uma solução para o caso de sequências dependentes dos tempos de *setup*, através da metaheurística colônia de formigas.

1.1.2 Objetivos específicos

Para atingir o objetivo geral seguirá-se os seguintes objetivos específicos:

- Adaptar um modelo para o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*;

- Testar a influência da consideração dos tempos de *setup*, na qualidade de soluções iniciais obtidas com heurísticas clássicas;

- Propôr um algoritmo baseado na metaheurística colônia de formigas, com considerações dos tempos de *setup* nas regras de transição;

2 Fundamentação teórica

Neste capítulo, apresenta-se a base teórica utilizada neste trabalho. Na Seção 2.1, introduz-se o planejamento e controle da produção (PCP), bem como, as decisões, os diferentes ambientes, medidas de desempenho e outros aspectos que podem ser considerados na programação da produção. Na Seção 2.2, apresenta-se a motivação para a escolha do problema estudado, por meio de um levantamento de publicações em torno do problema. Após, na Seção 2.3 apresenta-se um modelo já proposto na literatura para o problema de *flow shop* e, na Seção 2.4, apresenta-se alguns métodos já consagrados na literatura para gerar sequências de tarefas que, serão utilizados como base para, a geração de sequências iniciais na resolução do problema proposto neste trabalho. Para encerrar o capítulo, na Seção 2.5, é feito um levantamento acerca da metaheurística colônia de formigas, comenta-se sobre os principais experimentos que contribuíram para a criação desta metaheurística bem como, os principais algoritmos baseados nesta técnica de otimização.

2.1 Programação da produção

O sistema produtivo transforma entradas (recursos) em saídas (produtos). Pode-se dividir um sistema de produção em três partes: estrutural (parte física da empresa), social (relações sociais entre as pessoas da empresa) e organizacional (gerenciamento do processo produtivo).

O planejamento e controle da produção (PCP) é um apoio para o sistema produtivo, as decisões com relação ao PCP são tomadas no nível organizacional da empresa. O sistema produtivo precisa ser pensado em termos de prazos, pode-se dividir o horizonte de planejamento em três níveis: longo, médio e curto prazo, salienta-se que as atividades do PCP são importantes nestes três níveis (TUBINO, 2009).

A longo prazo, tem-se o nível estratégico, chamado plano de produção, cuja finalidade é definir a capacidade de produção do sistema de modo a atender os clientes, baseia-se na previsão de vendas a longo prazo. A médio prazo, tem-se o nível tático, chamado plano mestre de produção (PMP), neste nível define-se a forma de operação da empresa de modo a usar eficientemente a capacidade para atender os pedidos. E a curto prazo, tem-se o nível operacional, chamado de programação da produção, nesse nível administra-se estoques, otimiza-se o uso de recursos, organiza-se a sequência de tarefas, o uso das máquinas, a escala dos funcionários a entrega dos produtos, etc. (TUBINO, 2009).

A pesquisa operacional oferece modelos e métodos de solução para diferentes problemas encontrados na programação da produção. A teoria de programação (*Scheduling*), que trata dos problemas de otimização combinatória subjacentes aos problemas reais de programação de operações, inclui, além dos problemas de programação da produção em fábricas, problemas de

programação de entregas (ou roteirização de veículos), programação de escalas de trabalho, ente muitos outros (LUSTOSA et al., 2008).

Os objetivos da programação da produção podem ser divididos em três classes: cumprimento de prazos de entrega, tempo de fabricação ou processamento das tarefas e a minimização dos custos de produção. Assim, percebe-se que a programação da produção é uma atividade operacional que envolve decisões a serem tomadas para um horizonte de tempo limitado: horas, dias ou semanas. Apresenta-se a seguir uma classificação de possíveis decisões a serem tomadas, para a programação da produção, essas decisões são apresentadas por Lustosa et al. (2008).

Designação (*assignment*): Determinação de onde (ou por quem) a tarefa será executada.

Sequenciamento (*sequencing*): Determinação da sequência em que as tarefas devem ser executadas em um dado recurso.

Programação (*scheduling*): Determinação de quando a tarefa deverá ser iniciada e terminada, não apenas a sequência.

Despacho (*dispatching*): Quando e para quem (ou centro de trabalho) a tarefa deverá ser emitida (ou liberada).

Controle (*control*): Acompanhamento das tarefas para garantir que o programa se mantenha válido e determinação de eventuais necessidades de intervenção ou reprogramação.

Apressamento (*expediting*): Acelerar uma tarefa aumentando sua prioridade para manter o programa válido ou para atender uma necessidade específica.

Carregamento de oficinas (*shop loading*): Definição dos roteiros e programação das tarefas dentro de uma fábrica, ou seção de um sistema produtivo.

Em um sistema produtivo, pode-se pensar em diferentes ambientes de produção, de acordo com o número de máquinas, padrão de fluxo das tarefas, controle das máquinas por computadores ou por pessoas. Na teoria de programação, quanto as máquinas, considera-se quatro ambientes básicos: uma máquina (*single machine shop*), máquinas em paralelo (*parallel machine shop*), *job shop* e *flow shop*. A seguir apresenta-se esses ambientes básicos de programação segundo Lustosa et al. (2008) e Arenales et al. (2007).

Máquina única (*single machine shop*): considera-se a existência de n tarefas a serem processadas em uma única máquina, cada tarefa possui parâmetros específicos como: tempo de *setup*, tempo de produção, data de entrega, data de liberação etc. Embora pareça pouco representativo de uma situação real, o problema de programação de uma única máquina permite apresentar fundamentos de programação que serão úteis na solução de problemas mais complexos. Além disso, em alguns casos, a programação da fábrica pode estar subordinada à programação de uma máquina principal, ou outra situação interessante ocorre em processos de produção contínua em que uma linha inteira, ou até mesmo a fábrica (ou usina), pode ser considerada como uma única grande máquina.

Máquinas em paralelo (*parallel machine shop*): considera-se *n* tarefas que devem ser processadas em *m* máquinas semelhantes. Nesse problema, deve-se determinar quais as tarefas serão alocadas a cada máquina e, para cada uma, qual a sequência de processamento. O problema pode ser ainda generalizado, considerando-se que as máquinas não são idênticas, situação em que os tempos de operação podem variar conforme a máquina.

Job shop: considera-se *n* ordens que devem ser processadas em *m* máquinas de acordo com um roteiro pré-estabelecido, cada tarefa tem um tempo de operação específico em cada uma das máquinas.

Flow shop: é um caso particular do *job shop*, considera-se *n* tarefas, que devem ser processadas sequencialmente nas *m* máquinas, com o mesmo roteiro. Cada tarefa tem um tempo de operação específico em cada uma das máquinas. Dentro desse ambiente, podem-se classificar uma situação particular, se a sequência de tarefas é a mesma em todas as máquinas, tem-se um ***flow shop* permutacional**.

A avaliação da qualidade de um programa de produção é feita por medidas de desempenho. Arenales et al. (2007) classifica que as principais medidas de desempenho são:

Tempo total (*Makespan*): instante de término de processamento de todas as tarefas, que é equivalente, ao instante de término da última tarefa, na última máquina.

Tempo de fluxo total (*total flow time*): soma dos instantes de término de processamento das tarefas, esta medida relaciona o estoque em processamento.

Atraso (*tardiness*): pode-se considerar um atraso máximo ou um atraso total, esta medida relaciona-se com a data de entrega das tarefas.

Pontualidade (*lateness*): diferença entre a data de conclusão e a data prometida da tarefa. Se positiva, representa um atraso, se negativa, um adiantamento.

Número de tarefas atrasadas: quantidade de tarefas (ou ordens) concluídas após a data de entrega prometida.

Lustosa et al. (2008) ressalta que pode-se utilizar medidas de desempenho para avaliar a utilização dos recursos, com base em um único recurso ou a todos os recursos e também alguns aspectos adicionais devem ser considerados na programação da produção, entre eles destaca-se os seguintes:

Utilização (*utilization*): razão entre o tempo em que o recurso foi efetivamente utilizado com relação ao tempo total disponível desse recurso.

Ociosidade (*idleness*): medida oposta a utilização, mede quanto tempo o recurso ficou ocioso, ou seja, sem uso.

Tempo de preparação dependente da sequência (*sequence dependent setup*): situação em que o tempo de preparação da máquina muda em função da sequência de execução das ordens.

Preempção (*preemption*): situação em que se admite a interrupção de uma tarefa para processamento de outra. Nos modelos, consideram-se dois tipos de interrupção: a preempção com recomeço (quando a tarefa interrompida tem seu processamento anterior perdido) e a preempção com continuação (quando a tarefa interrompida é retomada sem prejuízo do tempo anterior).

Arenales et al. (2007) acrescenta outros aspectos que podem ser observados na programação da produção como, tarefas que esperam para serem processadas em uma máquina, situam-se em geral, em um *buffer*. Pode-se considerar dois casos em que não permite-se *buffers*. Quando o término da operação de uma tarefa em uma máquina, bloqueia (*blocking*) esta máquina, até que, a máquina seguinte esteja disponível para o processamento, ou no caso sem espera (*no-wait*), em que as tarefas devem ser processadas continuadamente em todas as máquinas, sem interrupção.

Neste trabalho, estuda-se uma das variantes do problema de *flow shop*, considerando a sequência de tarefas dependente dos tempos de *setup*. Na próxima seção, apresenta-se um levantamento de publicações em torno do problema, o que nos motivou a estudá-lo.

2.2 Motivação

O problema de *flow shop* permutacional é um problema comum, em indústrias que precisam de um sequenciamento de tarefas eficiente, de modo a ajudar a empresa atingir algum objetivo. Nas últimas décadas, o problema foi tema de estudo de muitos pesquisadores, porém, este estudo considerando a sequência dependente dos tempos de *setup*, é relativamente mais recente. Neufeld, Gupta e Buscher (2016) fizeram um levantamento das publicações acerca do problema de *flow shop* desde 1976, mostraram que o número de publicações considerando sequências dependentes dos tempos de *setup* cresceu após 2009.

Inicialmente, as soluções propostas envolviam técnicas de enumeração como, o método *branch and bound*. Por se tratar de um problema NP-*hard*, o custo computacional envolvido é alto, portanto, esses métodos resolvem apenas problemas de tamanhos pequenos a moderados. Para resolver problemas de tamanhos maiores, pesquisadores vêm propondo soluções por meio de metaheurísticas e, por heurísticas construtivas e de melhoria.

Johnson (1954) propôs um algoritmo para o problema de *flow shop* de n tarefas mas, somente para 2 máquinas, com o objetivo de minimizar o *makespan*. Ignall e Schrage (1965) aplicaram o algoritmo *branch and bound* em um problema de *flow shop*. Ríos-Mercado e Bard (1999) resolveram o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, usando um algoritmo *branch and bound*, com o objetivo de minimizar o *makespan*. Danneberg, Tautenhahn e Werner (1999) compararam o desempenho de diferentes

algoritmos heurísticos, para problemas de programação *flow shop* com tempos de *setup* e, tamanhos limitados de lotes.

Ying e Liao (2004) afirmaram serem os primeiros a aplicar o sistema de colônia de formigas (*Ant Colony System* - ACS), proposto por Dorigo e Gambardella (1997), em um problema de *flow shop* permutacional, sem considerar os tempos de *setup*, com o objetivo de obter um sequenciamento de tarefas que minimize o *makespan*. Compararam a eficiência do algoritmo ACS com algoritmo genético, *simulated annealing* e busca local, constataram que o ACS obteve resultados superiores para este problema.

Rajendran e Ziegler (2004) resolveram o problema de *flow shop* permutacional, sem considerar os tempos de *setup*, com três propostas diferentes, uma baseada em um algoritmo já existente com algumas modificações e, outras duas são propostas novas. Primeiramente, utilizou-se os algoritmos com o objetivo de minimizar *makespan*, depois trocou-se o objetivo para minimizar o tempo de *flow time* total. Concluíram que os algoritmos propostos por eles são superiores aos existentes, quando comparado em instâncias já consagradas.

Gajpal, Rajendran e Ziegler (2006) resolveram o problema de *flow shop* permutacional, considerando a sequência de tarefas dependente dos tempos de *setup*. Propuseram um novo algoritmo, chamado PACA, baseado na otimização colônia de formigas (ACO), inspirado no algoritmo proposto por Rajendran e Ziegler (2004) em termos de procedimentos de busca local, geração de sequência inicial, inicialização e atualização dos feromônios. Na Subseção 2.4.4 apresenta-se mais detalhes a respeito do algoritmo PACA.

Luo e Chu (2007) também desenvolveram um algoritmo *Branch and Bound*, para a programação de uma única máquina, com tempos de preparação dependentes da sequência, com o objetivo de minimizar o atraso (*tardiness*).

Allahverdi et al. (2008) fizeram um levantamento de publicações que consideravam problemas de programação (*scheduling*), com tempos de *setup* dependentes e independentes da sequência de tarefas, separados em lotes ou não. Apresentaram onze trabalhos que estudaram as características ou, resolveram o problema de *flow shop* permutacional, considerando sequência dependente dos tempos de *setup*, para minimizar o *makespan*. Para resolução do problema, os trabalhos apresentaram diferentes métodos: *Branch and Bound*, *Branch and Cut*, Busca Tabu, heurística baseada no caixeiro viajante, programação linear inteira mista, Algoritmo Genético, algoritmo heurístico baseado em penalidade e *Iterated Greedy*.

Yagmahan e Yenisey (2008) resolveram um problema de *flow shop* permutacional, sem considerar os tempos de *setup*, utilizando ACO (*ant colony optimization*), com multi-objetivos: minimizar o *makespan*, o tempo total de fluxo (total *flow time*) e o tempo total ocioso das máquinas (total *machine idle time*). Após, Yagmahan e Yenisey (2010) resolveram o problema de *flow shop* permutacional, sem considerar os tempos de *setup*, utilizando ACS (*ant colony system*) com multi-objetivos de, minimizar o *makespan* e o tempo total de fluxo (total *flow time*).

Souza (2009) resolveu o problema de *flow shop* com *m* máquinas e *n* tarefas, sem a obrigatoriedade de passar por todas as máquinas, considerou punições para atrasos e adiantamentos na conclusão das tarefas. Propôs um modelo matemático que foi resolvido utilizando o CPLEX e, soluções heurísticas e combinadas, encontradas a partir de soluções iniciais, geradas pelas heurísticas, posteriormente foram utilizadas no modelo. Branco (2011) propôs um novo método heurístico construtivo, com objetivo de minimizar o *makespan*, em um problema de programação *no-idle flow shop* permutacional, isto é, nenhuma máquina permite intervalo de tempo ocioso (inativo), entre operações consecutivas.

Mirabi (2011) resolveu o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, utilizando um algoritmo baseado na otimização colônia de formigas (ACO). Para inicialização dos feromônios, o autor usou soluções iniciais geradas com a heurística NEH modificada e com um método, proposto por ele, para calcular um peso para as arestas. Na Subseção 2.4.5 apresenta-se mais detalhes a respeito deste trabalho.

Ahmadizar (2012) propôs o algoritmo NACA (*new ant colony algorithm*), baseado na metaheurística colônia de formigas para resolver o problema de *flow shop* permutacional, sem considerar os tempos de *setup*, com o objetivo de minimizar o *makespan*. No algoritmo a inicialização das trilhas de feromônios é baseada numa sequência inicial e, as atualizações dos feromônios são limitadas inferiormente e superiormente por valores que mudam dinamicamente, levando em consideração o melhor *makespan* já encontrado. Depois de cada formiga construir uma solução, aplica-se um processo de busca local para melhorá-la, após todas as formigas construírem suas soluções, aplica-se uma atualização global privilegiando a melhor solução.

Li e Zhang (2012) resolveram o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, com algoritmos genéticos híbridos adaptados, com o objetivo de minimizar separadamente, o *makespan* e o atraso total ponderado. Vanchipura e Sridharan (2013), desenvolveram dois algoritmos heurísticos construtivos, para minimizar o *makespan*, em um ambiente *flow shop* permutacional, com sequência dependente dos tempos de *setup*. Saravanan, Vijayakumar e Srinivasan (2014) propuseram um algoritmo metaheurístico, *artificial immune system*, para resolver o problema de *flow shop* permutacional, com tempos de *setup* e, função objetivo minimizando simultaneamente *makespan*, *tardiness*, *earliness* e *total completion time*.

Mirabi (2014) desenvolveu algoritmo para o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, baseado no algoritmo genético, com objetivo de minimizar o *makespan*, desenvolveu um novo algoritmo genético híbrido (HGA), que aplica uma abordagem modificada para gerar a população de cromossomos iniciais, utiliza-se de três operadores genéticos para fazer uma nova população e, usa uma heurística de melhoria. Na comparação com três heurísticas, o algoritmo proposto mostrou ser muito competitivo.

Shen, Gupta e Buscher (2014) desenvolveram um algoritmo baseado na busca tabu, para resolver um problema de *flow shop batching*, isto é, separar famílias de produtos em lotes, com a sequência dependente dos tempos de *setup*, com o objetivo de minimizar o *makespan*.

Abordagens recentes do problema estudado neste trabalho, mostram propostas de resoluções por técnicas de enumeração, metaheurísticas, heurísticas de melhoria e construtivas. Porém, somente seis trabalhos (YING; LIAO, 2004), (GAJPAL; RAJENDRAN; ZIEGLER, 2006), (YAGMAHAN; YENISEY, 2008), (YAGMAHAN; YENISEY, 2010), (MIRABI, 2011) e (AHMADIZAR, 2012), utilizam a metaheurística colônia de formigas, sendo que, apenas em (GAJPAL; RAJENDRAN; ZIEGLER, 2006) e (MIRABI, 2011), leva-se em consideração sequência dependente dos tempos de *setup*. Porém, nestes trabalhos, não utiliza-se considerações explícitas, referente aos tempos de *setup*, nas regras de transição. Acredita-se que essa consideração seja de suma importância, visto que, as regras de transição são utilizadas para construir a sequência de tarefas.

Neste trabalho, propõe-se usar a metaheurística colônia de formigas, para resolver o problema de programação em um ambiente *flow shop* permutacional, com sequência dependente dos tempos de *setup*, utilizando informações referente a soma dos tempos de *setup*, nas regras de transição. Acredita-se que este trabalho seja o primeiro a fazer estas considerações, no Capítulo 4, mostra-se com detalhes essa proposta. A seguir, na Seção 2.3, apresenta-se um modelo já proposto na literatura, para o problema de *flow shop* (não permutacional) e, sem considerar os tempos de *setup*.

2.3 Problema de *flow shop*

Na Seção 2.1 introduziu-se alguns conceitos sobre a programação da produção, também definiu-se alguns diferentes ambientes de produção de acordo com o número ou disposição das máquinas, padrão de fluxo de tarefas e, dentre estes ambientes, definiu-se o problema de *flow shop*.

Arenales et al. (2007) define o ambiente de produção *flow shop* como um caso particular do ambiente *job shop*, em que as n tarefas têm o mesmo roteiro nas m máquinas e, quando a sequência de tarefas for a mesma em todas as máquinas têm-se um *flow shop* permutacional. A seguir apresenta-se a formulação proposta por eles para o problema de *flow shop* (não permutacional) e, sem considerar os tempos de *setup*.

Seja j o índice que indica a posição na sequência de tarefas e, C_{max} representa o instante de término da última tarefa, na última máquina (m).

Considere o parâmetro:

p_{ik} = tempo de processamento da tarefa i na máquina k, com $i =, 1, ..., n$ e $k = 1, ..., m$.

Defina as variáveis:

t_{kj} = instante de início de processamento da tarefa na posição j na máquina k

$$x_{ij} = \begin{cases} 1 & \text{se a tarefa } i \text{ é designada à posição } j \\ 0 & \text{caso contrário} \end{cases}$$

e considere o seguinte modelo:

$$min\ C_{max} = t_{mn} + \sum_{i=1}^{n} p_{im}\ x_{in} \tag{2.1}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad i = 1, ..., n \tag{2.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad j = 1, ..., n \tag{2.3}$$

$$t_{1j} + \sum_{i=1}^{n} p_{i1}\ x_{ij} = t_{1,j+1} \quad j = 1, ..., n-1 \tag{2.4}$$

$$t_{11} = 0 \tag{2.5}$$

$$t_{k1} + \sum_{i=1}^{n} p_{ik}\ x_{i1} = t_{k+1,1} \quad k = 1, ..., m-1 \tag{2.6}$$

$$t_{kj} + \sum_{i=1}^{n} p_{ik}\ x_{ij} \leq t_{k+1,j} \quad j = 2, ..., n \quad k = 1, ..., m-1 \tag{2.7}$$

$$t_{kj} + \sum_{i=1}^{n} p_{ik}\ x_{ij} \leq t_{k,j+1} \quad j = 1, ..., n-1 \quad k = 2, ..., m \tag{2.8}$$

$$c \in R_{+}^{nm} \quad x \in B^{nn} \tag{2.9}$$

A Equação (2.1) representa a função objetivo de minimizar o *makespan*. As restrições da Equação (2.2) garantem que cada tarefa i está associada a uma única posição e, as restrições da Equação (2.3) asseguram que cada posição j esta associada a uma única tarefa. Uma representação das restrições das Equações (2.4) e (2.5) pode ser visualizada na Figura 1.

Na Figura 1, considera-se um exemplo aleatório de cinco tarefas (T1, T2, T3, T4 e T5) que, estão sendo processadas na primeira máquina (M1). Supõe-se que a sequência T3 → T5 → T1 → T4 → T2, é a melhor sequência de processamento para a primeira máquina (M1). O índice j refere-se a posição, portanto na posição $j = 1$ tem-se T3, na posição $j = 2$ tem-se T5, e assim sucessivamente, até a posição $j = 5$ onde tem-se T2. De acordo com o modelo, p_{ik} refere-se ao

tempo de processamento da tarefa i (Ti), na máquina k (Mk) e, t_{jk} refere-se ao instante de início de processamento, da tarefa que está na posição j, na máquina k.

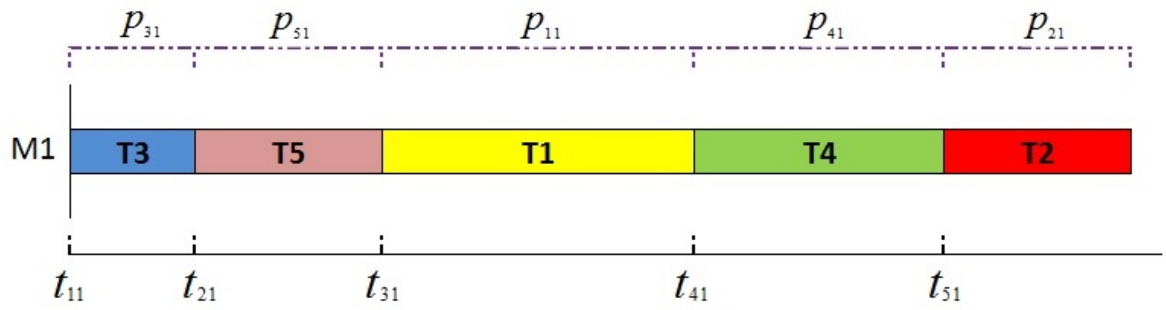

Figura 1 – Ilustração das restrições das Equações (2.4) e (2.5). Fonte: o autor.

As restrições da Equação (2.4) fazem com que o processamento de tarefas, na primeira máquina (M1), seja imediato, ou seja, a tarefa que está na posição j, deve ser processada imediatamente, após a tarefa que está na posição anterior, $j-1$. No exemplo ilustrado na Figura 1, evidencia-se esse processamento imediato. A restrição da Equação (2.5), estabelece que a primeira tarefa da sequência, comece com seu processamento na máquina 1, no instante 0.

A Figura 2 ilustra um caso aleatório, em que uma tarefa (T3), é processada como primeira tarefa da sequência (posição $j = 1$), em três máquinas (M1, M2 e M3). O processamento na segunda máquina (M2) inicia-se imediatamente após o fim do processamento da tarefa na primeira máquina (M1), ou seja, $t_{12} = p_{31}$. Bem como, o processamento na terceira máquina (M3), inicia-se imediatamente após o processamento da tarefa, na segunda máquina (M2) assim, $t_{13} = t_{12} + p_{32} = p_{31} + p_{32}$.

As restrições da Equação (2.6), garantem que a primeira tarefa da sequência, seja processada imediatamente na próxima máquina $k+1$, desde que seu processamento na máquina corrente k, tenha sido completado, a Figura 2 ilustra estas restrições.

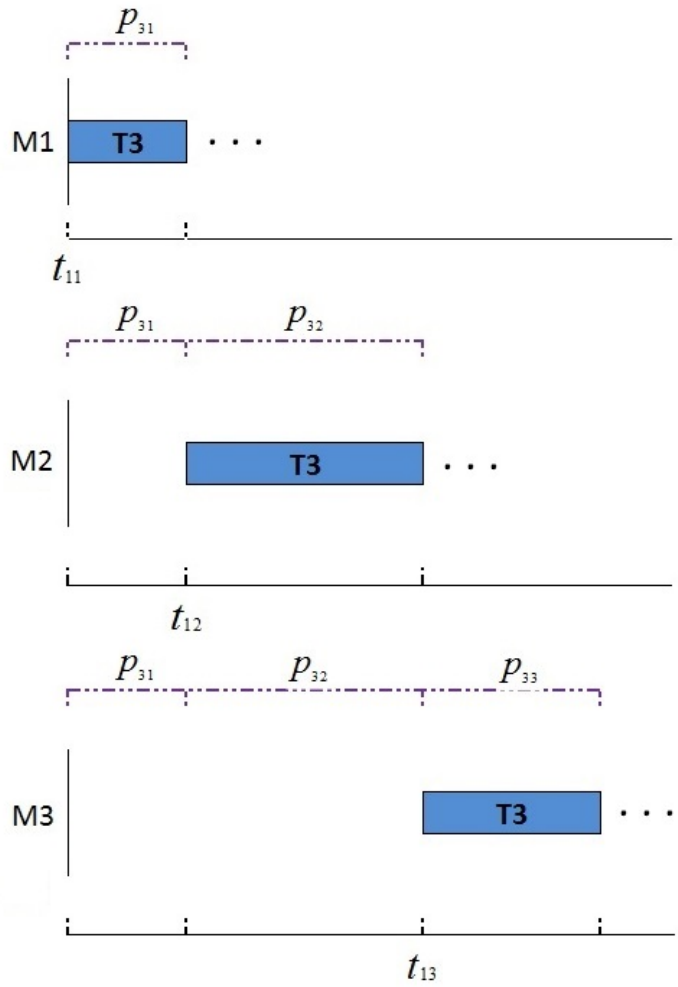

Figura 2 – Ilustração das restrições da Equação (2.6). Fonte: o autor.

A Figura 3, apresenta um caso aleatório de um problema de *flow shop* em que, cinco tarefas (T1, T2, T3, T4 e T5) estão sendo processadas em três máquinas (M1, M2 e M3), o fluxo de tarefas tem que seguir a ordem das máquinas M1 → M2 → M3 mas, a ordem de processamento das tarefas, não precisa ser a mesma em todas as máquinas e, nem todas as tarefas precisam passar em todas as máquinas. Se a sequência de tarefas, for a mesma em todas as máquinas, trata-se de um *flow shop* permutacional. Supõe-se que a sequência T3 → T5 → T1 → T4 → T2, é a melhor sequência de processamento na primeira máquina (M1), a sequência T3 → T5 → T1 → T4, é a melhor sequência de processamento na segunda máquina (M2) e, a sequência T3 → T5 → T1 → T2, é a melhor sequência de processamento na terceira máquina (M3).

As restrições da Equação (2.7), asseguram que uma tarefa na posição j, não pode ser iniciada na próxima máquina $k + 1$, antes do término do seu processamento na máquina corrente k. As restrições da Equação (2.8), garantem que uma tarefa na posição $j + 1$, não pode iniciar seu processamento na máquina k, antes que o processamento da tarefa na posição j, na mesma máquina k, tenha sido completado.

A ilustração das Equações (2.7) e (2.8) é apresentada na Figura 3, a tarefa T5, está na posição $j = 2$, na primeira e segunda máquina. Seu processamento na segunda máquina, inicia somente depois de concluir seu processamento na primeira máquina e, após a conclusão do processamento da tarefa T3, que está na posição $j = 1$. Na segunda máquina, tem-se um tempo de ociosidade entre o processamento de T5 → T1 e, T1 → T4 que, corresponde ao tempo de espera da máquina até a conclusão das tarefas T1 e T4, respectivamente, na primeira máquina. A restrição da Equação (2.9) indica o tipo das variáveis (ARENALES et al., 2007).

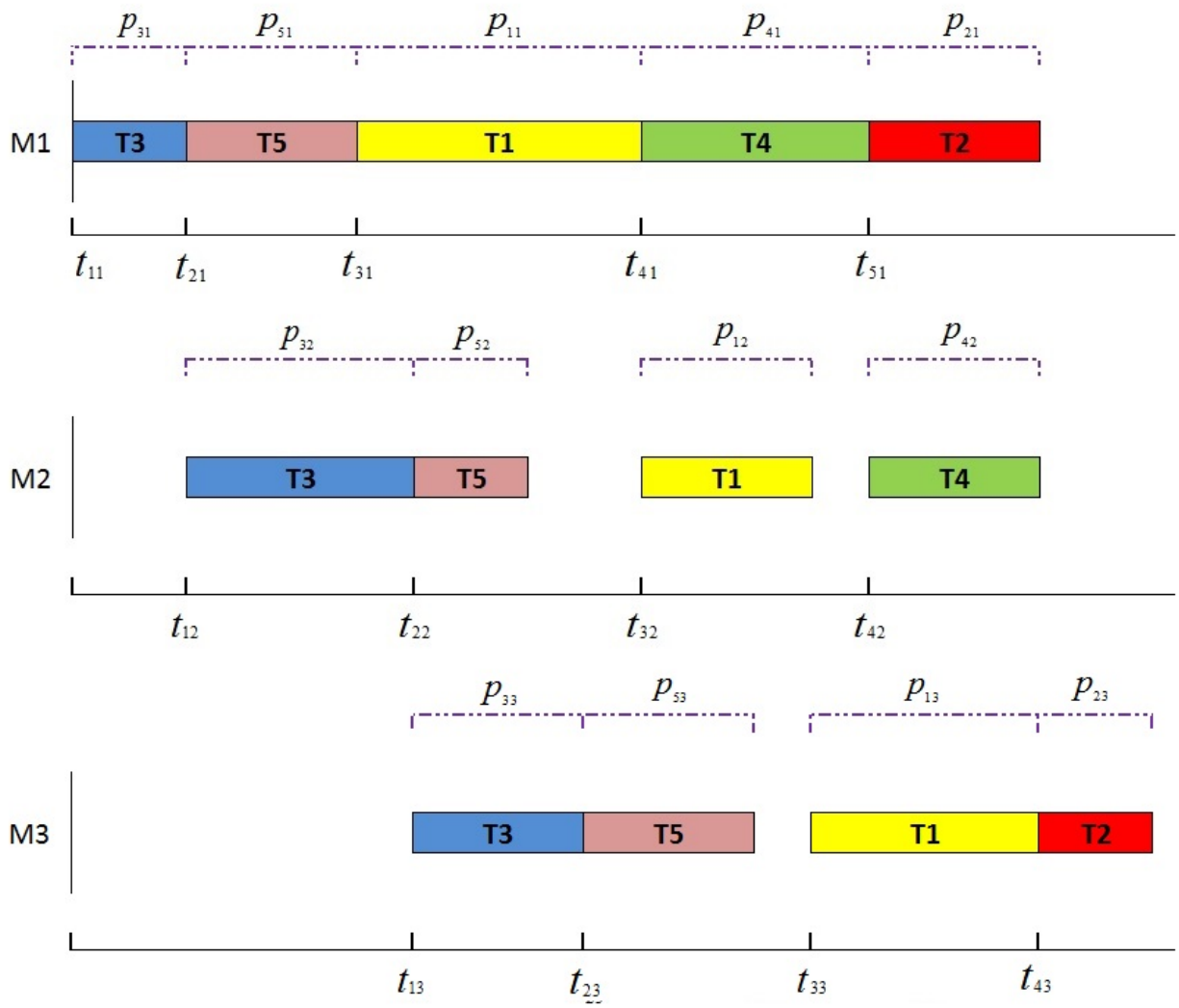

Figura 3 – Ilustração das restrições da Equação (2.7). Fonte: o autor.

O modelo apresentado, proposto por Arenales et al. (2007), é aplicado ao problema de *flow shop* que, difere do problema de *flow shop* permutacional, na obrigatoriedade da sequência de tarefas ser a mesma para todas as máquinas. A proposta deste trabalho é resolver o problema de *flow shop* permutacional, para isso, adaptou-se o modelo apresentado nas Equações (2.1) até (2.9) para o *flow shop* permutacional.

No Capítulo 3 apresenta-se um modelo (MFSP) para o *flow shop* permutacional que, foi formulado com base no modelo citado com dois diferenciais. Um deles é não associar a posição das tarefas nas variáveis de decisão e sim, associar a precedência das tarefas, ou seja, a variável $x_{ij} = 1$ representará que a tarefa j é precedida imediatamente pela tarefa i. Também, faz-se considerações a respeito dos tempos de *setup* (ou preparação de máquina) que, na prática, são utilizados para preparar a máquina para produzir um certo item, após ter terminado a produção de outro item. Dependendo do ramo de produção, os tempos de *setup* são insignificantes e, podem ser considerados junto com os tempos de processamento porém, em alguns casos, os tempos de *setup* podem alterar significativamente, dependendo da sequência de itens a serem produzidos.

A seguir, no Seção 2.4 apresenta-se alguns métodos de solução propostos na literatura, para o problema estudado neste trabalho. A importância de estudar estes métodos, vêm da necessidade de gerar soluções iniciais.

2.4 Métodos de solução propostos na literatura para o problema de *flow shop*

Nesta seção, apresenta-se alguns métodos já consagrados na literatura para gerar sequências de tarefas. Esses métodos foram a base para a geração de sequências iniciais deste trabalho. Na Subseção 2.4.1, exibe-se um índice, proposto por Palmer (1965), para ordenar sequências de tarefas com base nos tempos de processamento.

Na Subseção 2.4.2, apresenta-se a Heurística NEH, proposta por Nawaz, Enscore e Ham (1983), especialmente para gerar sequências de tarefas para problemas de *flow shop* permutacional. Após, na Subseção 2.4.3, expõe-se uma modificação para Heurística NEH, proposta por Ruiz, Maroto e Alcaraz (2005).

Em seguida, na Subseção 2.4.4, exibe-se o algoritmo PACA, proposto por Gajpal, Rajendran e Ziegler (2006), baseado na metaheurística colônia de formigas para resolução de um problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*.

Na Subseção 2.4.5, apresenta-se um método que calcula peso para as arestas, levando em consideração, os tempos de processamento e o tempo de *setup* inicial (tempo de preparo da máquina caso a tarefa seja a primeira da sequência). Este método foi proposto por Mirabi (2011), como parte de um algoritmo que utiliza a metaheurística colônia de formigas para resolução de um problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*.

2.4.1 Heurística de Palmer

Palmer (1965) propôs o cálculo de um índice para ordenar a sequência de tarefas com base nos tempos de processamento, esse índice foi chamado por Palmer de *slope index*. Ele prioriza tarefas, com tempos de processamento, que tendem a diminuir de máquina para máquina.

O *slope index* para a tarefa i é denotado por Q_i e calculado por (2.10), onde p_{ik} é o tempo de processamento da tarefa i na máquina k com $k = 1, ..., m$.

$$Q_i = \sum_{k=1}^{m} (2k - m - 1)p_{ik} \quad \forall i = 1, \ldots, n \tag{2.10}$$

Depois de calculado o *slope index* de todas as tarefas, constrói-se uma sequência decrescente sendo que, a prioridade das tarefas segue a ordem desta sequência. Os maiores valores de Q_i, estão associados as tarefas i, que possuem menores tempos de processamento nas primeiras máquinas e, maiores tempos de processamento nas últimas máquinas. A seguir, apresenta-se uma ilustração numérica da heurística de Palmer (PALMER, 1965).

2.4.1.1 Ilustração numérica da heurística de Palmer

Apresenta-se uma ilustração numérica da heurística de Palmer, com intuito de ajudar os leitores no entendimento da mesma. Considere um problema de*flow shop* com 4 tarefas ($n = 4$) e 5 máquinas ($m = 5$).

Tempos de processamento p_{ik}

		Máquinas (k)				
		1	**2**	**3**	**4**	**5**
	1	5	9	8	10	1
Tarefas	**2**	9	3	10	1	8
(i)	**3**	9	4	5	8	6
	4	4	8	8	7	2

Calcular $Q_i = \sum_{k=1}^{5} (2k - m - 1)p_{ik}$

$Q_1 = (2-5-1)5+(4-5-1)9+(6-5-1)8+(8-5-1)10+(10-5-1)1 = -20-18+0+20+4 = -14$

$Q_2 = (2-5-1)9+(4-5-1)3+(6-5-1)10+(8-5-1)1+(10-5-1)8 = -36-6+0+2+16 = -24$

$Q_3 = (2-5-1)9+(4-5-1)4+(6-5-1)5+(8-5-1)8+(10-5-1)6 = -36-8+0+16+12 = -16$

$Q_4 = (2-5-1)4+(4-5-1)8+(6-5-1)8+(8-5-1)7+(10-5-1)2 = -16-16+0+14+4 = -14$

Organizar as tarefas em ordem decrescente de Q_i.

Observa-se que as tarefas 1 e 4 tiveram o mesmo valor de Q_1. Ao programar este método, deve-se ter um critério de escolha caso aconteça esta situação. Aqui, como trata-se de uma ilustração, iremos calcular o *makespan* das duas.

Sequência 1-4-3-2: *Makespan* = 61

Sequência 4-1-3-2: *Makespan* = 61

Evidencia-se que, coincidentemente as duas sequências produzem o mesmo valor para o *makespan*, então é indiferente, usar qualquer uma delas. Na próxima subseção, apresenta-se a heurística NEH proposta por Nawaz, Enscore e Ham (1983).

2.4.2 Heurística NEH

A heurística NEH proposta por Nawaz, Enscore e Ham (1983), foi desenvolvida para problemas de *flow shop* permutacional, com intuito de dar prioridade a tarefas com tempo total de processamento maior. A ideia central dessa heurística construtiva é, dar prioridade a tarefas com maior tempo total de processamento, em todas as máquinas.

O algoritmo proposto é apresentado a seguir no Algoritmo 1. Considere n o número de tarefas, m o número de máquinas, p_{ik} o tempo de processamento da tarefa i na máquina k e w representa a posição da sequência de tarefas, onde $1 \leq w \leq n$.

Algoritmo 1: Heuristica NEH (NAWAZ; ENSCORE; HAM, 1983).

Entrada: Número de tarefas (n), número máquinas (m) e tempos de processamento (p_{ik});
início
  para $i = 1$ **até** n **faça**
    $T_i = \sum_{k=1}^{m} p_{ik}$
  fim
  Organizar uma sequência de tarefas de acordo com a ordem decrescente dos valores de T_i;
  Pegar as tarefas, da primeira e segunda posição ($w = 1$ e $w = 2$), da sequência construída no passo anterior;
  Calcular o *makespan*, das duas possíveis sequências, formadas somente com as duas tarefas do passo anterior;
  para $w = 3$ **até** n **faça**
    Pegar a tarefa da w-ésima posição.
    Encontrar a melhor sequência, usando somente as w tarefas, sem alterar a ordem de prioridade das $(w - 1)$ tarefas já designadas.
    O número de enumerações neste passo é igual a w.
    se $w = n$ **então**
      Pare;
    fim
    senão
      $w = w + 1$
    fim
  fim
fim
Saída: Sequência de tarefas;

A seguir, apresenta-se uma ilustração numérica da heurística NEH para um problema de *flow shop* com 4 tarefas e 5 máquinas.

2.4.2.1 Ilustração numérica da heurística NEH

Nawaz, Enscore e Ham (1983) quando apresentaram a heurística NEH, fizeram uma ilustração numérica para ajudar aos leitores no entendimento da mesma. Neste trabalho, com o mesmo intuito, a seguir, apresenta-se uma ilustração numérica da heurística NEH. Considere um problema de *flow shop* com 4 tarefas e 5 máquinas.

Tempos de processamento p_{ik}

		Máquinas (m)				
		1	**2**	**3**	**4**	**5**
	1	5	9	8	10	1
Tarefas	**2**	9	3	10	1	8
(n)	**3**	9	4	5	8	6
	4	4	8	8	7	2

Calcular $T_i = \sum_{k=1}^{5} p_{ik}$

$T_1 = 5 + 9 + 8 + 10 + 1 = 33$

$T_2 = 9 + 3 + 10 + 1 + 8 = 31$

$T_3 = 9 + 4 + 5 + 8 + 6 = 32$

$T_4 = 4 + 8 + 8 + 7 + 2 = 29$

Organizar as tarefas em ordem decrescente de T_i.

Sequência = 1, 3, 2, 4 .

Encontrar a melhor sequência com as tarefas da primeira e da segunda posição, tarefa 1 e 3, da sequência do passo anterior.

Sequência 1-3: *Makespan* = 46

Sequência **3-1**: *Makespan* = **42**

A partir deste passo mantem-se a ordem de prioridade de tarefas: 3-1.

Posição $w = 3$.

Encontrar a melhor sequência com a tarefa na posição $w = 3$ (tarefa 2).

Os *makespans* das sequências parciais serão:

Sequência **3-1-2**: *Makespan* = **50**

Sequência 3-2-1: *Makespan* = 51

Sequência 2-3-1: *Makespan* = 51

A partir deste passo mantem-se a ordem de prioridade de tarefas: 3-1-2.

Posição $w = 4$.
Encontrar a melhor sequência com a tarefa na posição $w = 4$ (tarefa 4).
Os *makespans* das sequências parciais serão:
Sequência 3-1-2-4: *Makespan* = 58
Sequência 3-1-4-2: *Makespan* = 58
Sequência 3-4-1-2: *Makespan* = 57
Sequência **4-3-1-2**: *Makespan* = **54**
A partir deste passo mantem-se a ordem de prioridade de tarefas: 4-3-1-2.

Como $w = 4 = n$ então PARE.
Portanto, a sequência construída com a heurística NEH é: 4-3-1-2.

Evidencia-se que essa heurística, visa designar primeiro, as tarefas que possuem tempo total de processamento (T_i) maior. Na próxima subseção, apresenta-se uma modificação para a heurística NEH, proposta por Ruiz, Maroto e Alcaraz (2005).

2.4.3 Heurística NEH Modificada

Ruiz, Maroto e Alcaraz (2005) usaram a heurística NEH, com uma modificação para gerar diferentes indivíduos de uma população inicial, para o Algoritmo Genético, usado para resolver um problema de *flow shop* permutacional.

A modificação proposta foi, após ordenar as tarefas por ordem de seus tempos de processamento (ordenar pelo valor decrescente de T_i), escolher duas tarefas aleatórias da lista ordenada, trocar com as duas primeiras tarefas e, então prosseguir com o restante da heurística NEH.

O fato de trocar as duas primeiras tarefas da sequência por, duas tarefas escolhidas aleatoriamente, resulta em uma sequência de tarefas, vizinha da primeira sequência já formada. Se esse processo for repetido várias vezes, funciona como um "gerador de vizinhos". Portanto, ao aplicar repetidamente essa heurística NEH modificada, pode-se obter vários indivíduos bons para formar uma população inicial. Os próximos dois métodos, apresentados nas subseções a seguir, são baseados na metaheurística colônia de formigas, que será melhor discutida na Seção 2.5.

2.4.4 PACA

Gajpal, Rajendran e Ziegler (2006) propuseram um novo algoritmo, chamado PACA, com base na metaheurística colônia de formigas, utilizado para resolver o problema de *flow shop*

permutacional, com sequência dependente dos tempos de *setup*. O algoritmo proposto é uma modificação do que propôs Rajendran e Ziegler (2004), para resolver o problema de *flow shop* permutacional, sem considerar os tempos de *setup*. A modificação é em termos de: inicialização e atualização de feromônios, procedimentos de busca local e geração de sequências iniciais. O algoritmo proposto é apresentado no Algoritmo 2.

Algoritmo 2: Algoritmo PACA (GAJPAL; RAJENDRAN; ZIEGLER, 2006)

Entrada: Número de tarefas (n), de máquinas (m), de formigas (f), tempos de processamento e *setup*, parâmetros(ρ, dif);

início

Obter uma sequência inicial;

Melhorar a sequência inicial por três procedimentos de busca local;

Calcular o *makespan* desta sequência inicial;

M_{best} será o *makespan* da sequência inicial;

Inicializar as trilhas de feromônios fazendo:

$\tau_{ip} = \frac{1}{M_{best}}$ onde τ_{ip} é o valor do feromônio da tarefa i que esta na posição p;

enquanto *condições de parada não forem atendidas* **faça**

Para cada formiga construa uma sequência de tarefas com base em uma regra de transição;

Se a sequência gerada for a mesma de uma formiga anterior, gerar outra sequência até que se obtenha uma sequência ainda não gerada por outra formiga;

Melhorar a sequência com a inserção de uma tarefa aleatória em todas as possíveis posições procurando diminuir o valor do *makespan*;

Calcular o *makespan* desta sequência M_{atual};

Comparar a solução encontrada com a melhor solução obtida até agora e atualizá-la se necessário;

Atualizar as trilhas de feromônios avaliando:

se *sequência encontrada é melhor que a melhor sequência já encontrada* **então**

$\tau_{ip}^{novo} = \rho\, \tau_{ip}^{velho} + \frac{1}{dif\ M_{atual}}$

fim

senão

$\tau_{ip}^{novo} = \rho\, \tau_{ip}^{velho}$

fim

fim

Aplicar três procedimentos de busca local na melhor sequência encontrada;

fim

Saída: Melhor sequência de tarefas;

Para a geração da sequência inicial, os autores utilizaram uma regra que chamaram de relação de preferência heurística. Esta relação de preferência heurística, identifica dentre as tarefas ainda não programadas, a que possui a menor soma de tempo ocioso (*idle time*) de máquina, ou seja, a tarefa que se inserida na sequência, renderá a menor soma de tempos de máquinas paradas. Insere-se a primeira tarefa e inicia-se a construção de uma sequência parcial. Após isso, quando identifica-se uma tarefa, ainda não sequenciada, com a menor soma de tempo ocioso, insere-se esta tarefa, em todas as possíveis posições da sequência parcial já construída, a escolha da melhor posição é com referencia a sequência que vai produzir o menor *makespan*.

Para a construção das sequências pelas formigas, utiliza-se regras de transição que, levam em consideração a sequência inicial gerada pela regra de preferência heurística já descrita e, o valor das trilhas de feromônios, portanto, explicitamente os tempos de *setup* não são levados em consideração. Porém, na atualização das trilhas de feromônios os tempos de *setup* são considerados implicitamente pois, as atualizações são feitas com informações acerca dos valores do *makespan* das sequências.

Na subseção a seguir, apresenta-se um outro método, proposto por Mirabi (2011), para resolver o problema de *flow shop* permutacional com sequência dependente dos tempos de *setup*.

2.4.5 Método de pesos

Mirabi (2011), resolveu o problema de *flow shop* permutacional com sequência dependente dos tempos de *setup*, utilizando um algoritmo baseado na otimização colônia de formigas. Para inicialização dos feromônios, o autor usou soluções iniciais geradas com a heurística NEH modificada e com um método que calcula um peso para cada aresta, levando em consideração os tempos de processamento e o tempo de *setup* inicial (tempo de preparo de máquina caso a tarefa seja a primeira da sequência). Esse método de pesos, também foi utilizado para gerar sequências iniciais para inicialização dos feromônios. No Algoritmo 3 apresenta-se o método de pesos proposto por Mirabi (2011).

A ideia deste método é calcular um peso z_i para cada tarefa i. Este peso carrega a soma dos tempos de *setup* inicial (soma dos tempos de *setup* para processar a tarefa i como primeira tarefa em todas as máquinas) e, o tempo médio de processamento da tarefa i em todas as m máquinas. Após calcular os z_i , calcula-se o peso w_{ij} de cada aresta (i, j). A sequência de tarefas é construída considerando os w_{ij} mínimos.

Algoritmo 3: Método de pesos proposto por Mirabi (2011).

Entrada: Número de soluções iniciais geradas com método de pesos (d), número de tarefas (n), número de máquinas (m), tempo de processamento (p_{ik} da tarefa i na máquina k) e tempos de *setup* ($s_{i_1 i_2 k}$ da tarefa i_1 para tarefa i_2 na máquina k);

início

  para *i=1* **até** n **faça**

    $z_i = \sum_{k=1}^{m} s_{0ik} + \frac{\sum_{k=1}^{m} p_{ik}}{m}$ onde s_{0ik} é o tempo de *setup* para processar a tarefa i como primeira tarefa na máquina k;

  fim

  para $i = 1$ **até** $n - 1$ **faça**

    para $j = 2$ **até** n **faça**

      se $i \neq j$ **então**

        $w_{ij} = z_j - z_i$

      fim

    fim

  fim

  para $a = 1$ **até** d **faça**

    Escolha aleatoriamente uma tarefa e coloque-a como primeira tarefa da sequência;

    para $b = 2$ **até** n **faça**

      Selecione uma tarefa j, ainda não sequenciada, que tenha w_{ij} mínimo e sequencie-a (onde i é a última tarefa que foi sequenciada);

    fim

  fim

fim

Saída: Conjunto com d sequências de tarefas.

O algoritmo baseado na metaheurística colônia de formigas, proposto para resolver o problema de *flow shop* permutacional com sequência dependente dos tempos de *setup*, utilizando esse método de pesos para gerar sequências iniciais e para a inicialização dos feromônios é apresentado a seguir em Algoritmo 4.

Algoritmo 4: Algoritmo proposto por (MIRABI, 2011)

Entrada: Número total de soluções iniciais (v), número de soluções iniciais com método peso (c), número de tarefas (n), de máquinas (m), de formigas (f), tempos de processamento e *setup*, parâmetros;

início

Obter v soluções iniciais sendo:

$v - c$ soluções usando a heurística NEH modificada;

c soluções usando o método pesos;

Inicializar os valores dos feromônios (da tarefa i para tarefa j) levando-se em consideração o valor do peso (w_{ij});

enquanto *condições de parada não forem atendidas* **faça**

Para cada formiga, construa uma sequência de tarefas, com base em uma regra de transição que, para ir da tarefa i para a tarefa j, leva em consideração o peso (w_{ij}) e o valor do feromônio (τ_{ij}).

Melhorar a sequência encontrada por três processos de busca local;

Atualizar os feromônios;

fim

fim

Saída: Sequências de tarefas.

Observa-se, no algoritmo acima, que na inicialização dos feromônios e nas regras de transição, leva-se em consideração os pesos w_{ij}, que por sua vez, carregam o tempo de *setup* inicial (s_{0ik}), caso a tarefa i seja a primeira tarefa em todas as máquinas. Porém, vale ressaltar que, para alguns segmentos produtivos, a consideração dos tempos de *setup* entre tarefas é muito importante, pois, dependendo da sequência, as máquinas podem precisar de um tempo muito maior de ajuste ou limpeza, para que possam processar a próxima tarefa.

Nesta seção apresentou-se alguns métodos, já presentes na literatura, para construção de sequências de tarefas. No Capítulo 4, propõe-se alguns métodos de solução, inspirados nestes recém apresentados, para o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*. A seguir na Seção 2.5, apresenta-se a metaheurística colônia de formigas que, posteriormente, será usada para resolver o problema de *flow shop* permutacional.

2.5 Metaheurística colônia de formigas

Algoritmos inspirados na natureza são comuns, inteligência coletiva ou de enxames (*swarm intelligence*), é um campo de pesquisa que, estuda algoritmos inspirados no comportamento de enxames (*swarm*), que são compostos por indivíduos autônomos simples que, cooperam com uma auto-organização, ou seja, os indivíduos são independentes, não seguem comando de um líder, eles controlam suas ações de modo que o enxame atinja seus objetivos. Os algoritmos de otimização colônia de formigas fazem parte deste campo de estudo.

Os algoritmos de otimização colônia de formigas, surgiram a partir da observação de experimentos feitos com formigas reais. Colônias de formigas reais, trabalham em conjunto para encontrarem alimentos e, se comunicam através do depósito no chão, de uma substância chamada feromônio. Assim, usando o mesmo conceito, pode-se ter colônias de formigas artificiais, que trabalham em conjunto para resolverem problemas de otimização e, podem se comunicar indiretamente, simulando feromônios artificiais.

Esta seção tem por objetivo fazer um levantamento teórico e histórico, de como surgiu a técnica de otimização colônia de formigas e, como foi proposta a metaheurística colônia de formigas. Na Subseção 2.5.1 comenta-se sobre o comportamento de formigas reais e, os principais experimentos que contribuíram para a criação da metaheurística colônia de formigas. Na Subseção 2.5.2 apresenta-se a metaheurística colônia de formigas e, na Subseção 2.5.3 os principais algoritmos baseados na otimização colônia de formigas.

2.5.1 Comportamento de formigas reais

Um dos primeiros pesquisadores a investigar o comportamento social dos insetos, foi o entomologista francês Pierre-Paul Grassé. Ele descobriu que alguns insetos são capazes de reagir a sinais que ativam uma reação, essas reações podem atuar como novos estímulos, tanto para o inseto como, para outros insetos da colônia. Grassé usou o termo estigmergia (*stigmergy*) para descrever esse tipo particular de comunicação (DORIGO; SOCHA, 2006).

Um exemplo de estigmergia é o movimento das formigas, enquanto elas caminham para chegar a uma fonte de alimento, depositam no chão uma substância chamada feromônio, as outras formigas são capazes de sentir o cheiro deste feromônio e, tendem a seguir caminhos com mais feromônio.

Goss et al. (1989) fizeram um experimento com formigas argentinas *Iridomyrmex humilis*, ligando o ninho a uma fonte de alimento, utilizando pontes com caminhos de tamanhos diferentes. Utilizou-se dois módulos da ponte mostrada na Figura 4-a, conforme mostrado na Figura 4-b. Cada módulo tem dois ramos de comprimentos diferentes e, cada ramo faz um ângulo de 30°, em relação ao eixo da ponte central. Cinco a dez minutos após a colocação da ponte, as formigas descobriram a comida e o tráfego na ponte aumentou, utilizando-se os dois ramos, alguns minutos mais tarde, tornou-se visível a utilização do caminho mais curto.

As primeiras formigas escolhem aleatoriamente o caminho a seguir, aquelas que vão pelo menor caminho, chegam antes das que vão pelo maior e, consequentemente, retornam mais rápido para o ninho. Assim, enquanto o caminho maior é marcado apenas pelas formigas que estão indo, o menor caminho é marcado em ambos os sentidos, por formigas indo e voltando. Portanto, o menor caminho acumula mais feromônios e, rapidamente, torna-se o preferido. Então, quanto maior a diferença de comprimento entre os caminhos, mais rápido as formigas convergem para o menor.

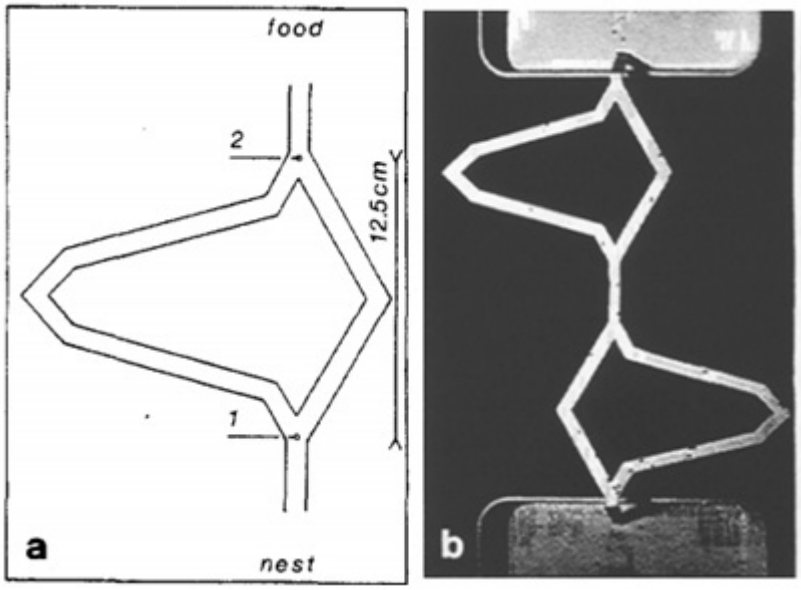

Figura 4 – Ponte com caminhos de tamanhos diferentes. Fonte: (GOSS et al., 1989)

Neste experimento, sabendo que as formigas argentinas *Iridomyrmex humilis* depositam feromônios durante a saída e a volta para o ninho, Goss et al. (1989) modelaram o comportamento das formigas ao atravessarem a ponte, considerando o depósito de uma unidade de feromônio por segundo. O modelo probabilístico que descreve o fenômeno descrito é apresentado na Equação (2.11), onde considera-se dois pontos $j = \{1, 2\}$ mostrados na Figura 4-a. Cada formiga escolhe o caminho curto com probabilidade p_{cj} e, o caminho longo com probabilidade p_{lj}, onde C_j é a quantidade de feromônios no caminho curto e, L_j no caminho longo.

A escolha do menor caminho em $j = 1$ (ou 2), faz com que a formiga chegue ao ponto oposto $j = 2$ (ou 1) k segundos mais tarde, onde k é obtido pela velocidade média da formiga e pelo comprimento do caminho. A escolha do maior caminho faz com que a formiga chegue ao ponto oposto kr segundos depois, onde r é a relação entre o comprimento dos dois caminhos e h é um parâmetro de ajuste.

$$p_{cj} = \frac{(C_j + k)^h}{(C_j + k)^h + (L_j + k)^h} \quad \text{onde} \quad p_{cj} + p_{lj} = 1 \tag{2.11}$$

O intervalo de tempo considerado nos experimentos foi de no máximo 30 minutos, que é o tempo médio de duração dos feromônios, portanto, nos modelos propostos, Goss et al. (1989) ignoraram a evaporação destes.

Deneubourg et al. (1990) também estudaram o comportamento, analisaram o padrão exploratório coletivo e realizaram experimentos com formigas argentinas *Iridomyrmex humilis*. Em um dos experimentos, o formigueiro era ligado a uma arena de 80x80 *cm*, por uma ponte binária simples conforme Figura 5. A arena era coberta com areia branca, ainda não explorada e, sem presença de fonte de alimento. Inicialmente a ponte não tinha feromônios. Ao caminhar as

formigas escolhiam aleatoriamente o caminho a seguir e depositavam feromônios, ao longo do tempo, uma das pontes ficava com mais feromônios até que toda a colônia convergia para aquela ponte.

Neste caso, conforme a Figura 5 e o experimento feito por Deneubourg et al. (1990), os dois caminhos têm o mesmo comprimento e, a chance inicial de utilizar qualquer um dos caminhos é a mesma. Com maior depósito de feromônio em um dos caminhos, ele passa a ser o preferido e então, as formigas seguem apenas um dos caminhos, dentre os dois possíveis.

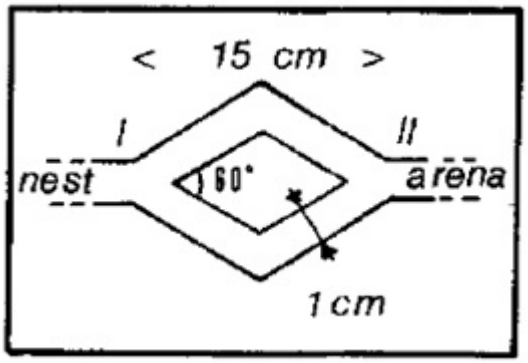

Figura 5 – Ponte Binária. Fonte: (DENEUBOURG et al., 1990)

O comportamento de formigas reais pode ser utilizado como inspiração para projetar formigas artificiais, assim, surgiu a ideia dos algoritmos de otimização colônia formigas que, inspiraram a criação da metaheurística colônia de formigas apresentada a seguir.

2.5.2 Otimização colônia de formigas

A otimização colônia de formigas (ACO), surgiu inspirada no comportamento biológico de formigas reais. Há muitas semelhanças entre colônias de formigas reais e artificiais, ambas são compostas por indivíduos que trabalham juntos para atingir um determinado objetivo. No caso de formigas reais, o problema consiste em encontrar o alimento enquanto que, no caso de formigas artificiais, o problema é encontrar uma solução para um problema de otimização. Nas formigas reais, a comunicação acontece através do depósito de feromônio no chão, as formigas artificiais podem simular o feromônio modificando valores de feromônios artificiais enquanto constroem soluções para o seu problema. Uma formiga sozinha (real ou artificial) é capaz de encontrar uma boa solução para o seu problema, mas apenas a cooperação entre os vários indivíduos permite encontrar melhores soluções.

Dorigo, Maniezzo e Colorni (1991) observaram a presença de efeitos sinérgicos (*synergetic*) na iteração das formigas, de fato, a qualidade da solução obtida aumenta quando o número de formigas que trabalham no problema aumenta. Eles também apresentaram a viabilidade de processos autocatalíticos (*autocatalytic*) como uma metodologia de otimização de aprendizagem. Definiram processo autocatalítico como um processo de *feedback* positivo, isto é, o processo se

reforça de uma maneira que faz com que a convergência seja muito rápida e, necessite de um mecanismo de limitação para não levar a explosão. A iteração de muitos processos autocatalíticos, pode levar a uma rápida convergência, para um subespaço do espaço de solução que contém soluções muito boas, em outras palavras, todas as formigas não convergem para uma única solução, mas para um subespaço de soluções, depois disso vão em busca de melhorar as soluções encontradas.

Dorigo, Caro e Gambardella (1999) ressaltam que ao usar autocatálise deve-se tomar cuidado para evitar a convergência prematura (estagnação), isto é, quando tem-se um ótimo local ou, por conta de condições iniciais que fazem com que um indivíduo não muito bom, seja melhor dentre todos os demais analisados, impedindo assim uma maior exploração do espaço de busca.

Em algoritmos de otimização colônia de formigas (ACO), um número finito de formigas artificiais, procura soluções de boa qualidade para o problema de otimização a ser resolvido. Cada formiga constrói uma solução, partindo de um estado inicial, selecionado de acordo com alguns critérios dependentes do problema. De acordo com a noção atribuída de vizinhança (dependente do problema), cada formiga constrói uma solução movendo-se através de uma sequência finita de estados vizinhos. Os movimentos são selecionados através da aplicação de uma regra que pode conter informações privadas da formiga (memória) ou, pelas trilhas de feromônios.

Segundo Dorigo, Caro e Gambardella (1999) a formiga pode carregar na memória informações para calcular o valor da solução gerada e, ou, a contribuição de cada movimento executado, em alguns problemas, por exemplo, alguns movimentos podem levar a formiga a soluções inviáveis, isso pode ser evitado através da exploração da memória da formiga. Formigas, portanto, podem construir soluções viáveis usando só o conhecimento sobre o estado local e, sobre os efeitos de ações que podem ser executadas no estado local. A informação pública local, pode ser uma informação heurística de um problema específico ou, informação codificada nas trilhas de feromônios, acumulada por todas as formigas desde o inicio do processo de pesquisa.

Dorigo e Caro (1999) formalizam a ideia de como uma colônia de formigas artificiais, pode encontrar boas soluções para problemas de otimização, representando o problema em um grafo, onde a tarefa principal das formigas artificiais será encontrar o caminho mais curto entre um par de nós do grafo. A Figura 6 apresenta traz o exemplo de um grafo onde, o caminho traçado em linha sólida representa um dos menores caminhos. As formigas que escolherem esse caminho vão chegar mais rápido ao destino, portanto, serão as primeiras a retornarem ao nó de origem.

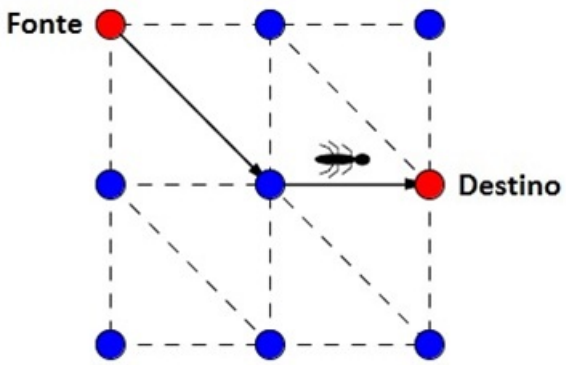

Figura 6 – Construção de soluções das formigas a partir de um nó fonte para um nó destino.

Quando considera-se um grafo, pode-se pensar em diferentes situações. No problema do caminho mínimo, citado acima, as formigas devem procurar o menor caminho entre dois pontos. Porém, pode-se pensar em outras situações, como o problema do caixeiro viajante, que consiste em determinar a menor rota, partindo de um nó de origem, para visitar todos os nós do grafo (visitando cada nó uma única vez) e, voltar ao nó de origem.

Dorigo e Socha (2006) enfatizaram a existência de algumas diferenças importantes entre formigas reais e artificiais: formigas artificiais vivem em um mundo discreto e se movem sequencialmente através de um conjunto finito de estados do problema, a atualização dos feromônios não é realizada da mesma maneira e também existem algumas implementações de formigas artificiais, que utilizam mecanismos adicionais, que não existem no caso das formigas reais, como por exemplo, uma busca local.

Quando projetam-se formigas artificiais, os feromônios podem ser liberados durante a construção de uma solução ou, depois da solução construída, movendo-se de volta por todos os pontos visitados. A decisão de como o feromônio será depositado depende das características do problema, em geral, a quantidade de feromônio depositada é feita proporcionalmente de acordo com a solução que a formiga construiu (ou está construindo), se a solução é de alta qualidade o depósito de feromônios pode ser maior.

A solução de um problema de otimização é expressa como uma solução de custo mínimo (ou menor caminho), de acordo com as restrições do problema. Soluções de alta qualidade são encontradas somente como o resultado da cooperação global, entre todos os agentes da colônia simultaneamente, construindo diferentes soluções. A seguir, apresenta-se os principais algoritmos para otimização colônia de formigas (ACO).

2.5.3 Algoritmos ACO

Os algoritmos baseados na ACO são populacionais e construtivos, tem-se uma população inicial de formigas e, a cada iteração cada formiga constrói uma solução para o problema. A primeira abordagem de um sistema baseado em ACO foi proposta por Dorigo, Maniezzo e

Colorni (1991), chamado de *Ant System* (AS), resultante de uma pesquisa sobre inteligência computacional e apresentado na Subseção 2.5.3.1.

Após, pesquisadores trabalharam para propor modificações no AS, assim surgiu o *Ant Colony System* (ACS), proposto por Dorigo e Gambardella (1997) e apresentado na Subseção 2.5.3.2 e, o *MAX-MIN Ant System* (MMAS) proposto por (STüTZLE; HOOS, 2000) e apresentado na Subseção 2.5.3.3.

A metaheurística otimização colônia de formigas (ACO) só foi definida e apresentada formalmente por Dorigo, Caro e Gambardella (1999). O algoritmo proposto pelos autores neste trabalho é apresentado no Algoritmo 5, com três laços, um define o critério de parada, outro limita a quantidade de formigas que serão criadas e o terceiro, constrói soluções. Entre dois dos laços há uma evaporação de feromônios. Além do algoritmo, que pode ser adaptado para diferentes tipos de problemas, o trabalho apresenta aplicações, que podem ser usadas com o algoritmo ACO e, reconhece os algoritmos AS e ACS, como derivados da metaheurística ACO.

Algoritmo 5: Algoritmo da metaheurística ACO, adaptado de Dorigo, Caro e Gambardella (1999).

```
início
    enquanto não atender as condições de parada faça
        enquanto tiver recursos disponíveis faça
            Criar uma nova formiga;
            Inicializar a formiga;
            enquanto estado atual ≠ estado destino faça
                Aplicar uma regra de transição para que a formiga mova-se para o próximo
                 estado;
                Depositar feromônios no arco visitado;
                Atualizar a memória da formiga;
            fim
        fim
        Evaporação de feromônios;
        Ações daemon (opcional);
    fim
fim
```

O número de formigas, pode ser definido de acordo com as características e necessidades do problema que será resolvido. Na construção das soluções com as regras de transição, as formigas podem levar em consideração informações do problema que está sendo resolvido e também, a quantidade de feromônios no caminho. As trilhas de feromônios podem ser inicializadas com valor nulo e atualizadas no decorrer do algoritmo ou, podem partir de valores iniciais utilizando-se de informações já conhecidas do problema, facilitando a convergência para a melhor solução.

O depósito ou evaporação dos feromônios consiste na atualização dos valores que, precisa ser equilibrada entre a exploração de novos pontos e, o conhecimento já acumulado de componentes específicos do problema que está sendo resolvido. Após a construção da solução, as formigas podem ser excluídas do sistema ou, podem construir novas soluções, partindo das trilhas de feromônios já atualizadas.

As ações *daemon* são feitas com base em informações globais recolhidas durante a resolução do problema, essas informações referem-se a observar o problema como um todo, por exemplo, após todas as formigas construírem suas soluções analisá-las e, fazer a ação de depositar feromônios em locais específicos, que produzam soluções melhores.

Diversas variantes da metaheurística otimização colônia de formigas foram propostas na literatura, Dorigo e Socha (2006) julgam que as três mais bem sucedidas são: *Ant System* (AS) (primeira implementação de um algoritmo ACO), *MAX-MIN Ant System* (MMAS) e *Ant Colony System* (ACS) que são apresentadas a seguir.

2.5.3.1 Ant System (AS)

Ant System (AS) foi a primeira abordagem da otimização colônia de formigas, proposta por Dorigo, Maniezzo e Colorni (1991) e posteriormente, aplicada no prolema clássico do caixeiro viajante por Dorigo, Maniezzo e Colorni (1996) que, sugeriram a aplicação em outros problemas de otimização como: problema do caixeiro viajante assimétrico e problemas de programação *job shop*.

No problema do caixeiro viajante, parte-se de um nó de origem para visitar todos os nós (ou cidades), visitando cada nó uma única vez e, volta-se a origem com a menor rota (ou menor custo) possível. Para construir soluções nos algoritmos baseados na otimização colônia de formigas (ACO), utiliza-se uma regra que chamamos de regra de transição.

No *Ant System* (AS), para construir uma solução, as formigas escolhem o melhor caminho a seguir com base no cálculo da probabilidade r_{ij}^{f}, da formiga f utilizar a aresta ij, conforme Equação (2.12), que é a regra de transição desta abordagem. Suponha que a formiga f está no nó i e deseja escolher um nó j para mover-se, Ω_f é um conjunto que contém todos os possíveis nós para ainda serem visitados pela formiga f, calcula-se o valor de r_{ij}^{f} para todo $j \in \Omega_f$ e move-se para o que nó j que resultar no maior r_{ij}^{f}.

Na Equação (2.12) τ_{ij} é a quantidade de feromônios na aresta ij, η_{ij} é o valor da informação heurística da aresta ij sendo que, os autores sugerem utilizar $\eta_{ij} = 1/d_{ij}$, onde d_{ij} é o

comprimento da aresta ij. Os parâmetros α e β são previamente definidos de modo a controlar a importância relativa dos feromônios e da informação heurística.

$$r_{ij}^{f} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\,\eta_{ij}^{\beta}}{\sum_{l\in\Omega_f}\tau_{il}^{\alpha}\,\eta_{il}^{\beta}} & \text{se } j \in \Omega_f \\ 0 & \text{caso contrário} \end{cases} \tag{2.12}$$

Quanto a inicialização dos feromônios, os autores sugerem definir pequenos valores aleatórios para todos os τ_{ij}. Em seus experimentos, os autores do AS utilizaram os mesmo valor para todas as arestas. O critério de parada sugerido por eles é, definir um número máximo de iterações. Já para a atualização de feromônios, são propostos três métodos diferentes: *ant-density*, *ant-quantity* e *ant-cycle* que serão apresentados nos Algoritmos 6, 7 e 8, respectivamente.

O Algoritmo 6 apresenta o AS com atualização de feromônios pelo método *ant-density*, esta atualização é feita após cada formiga completar uma solução, fixa-se um valor $\rho \in [0, 1]$, este valor determina a importância que será mantida dentre a quantidade de feromônios que havia na aresta (i, j) antes da atualização. Os parâmetros ρ e Q_1, devem ser fixados de acordo com o problema que será resolvido, dependendo dos valores atribuídos a estes parâmetros, a atualização faz com o que a quantidade de feromônios aumente com o aumento do valor de ρ e Q_1.

Algoritmo 6: *Ant System* (AS) com atualização de feromônios *ant-density*, adaptado de Dorigo, Maniezzo e Colorni (1991).

Entrada: número máximo de iterações (it_{max}), número de formigas (m), número de nós que cada formiga deve percorrer para completar uma solução, comprimento das arestas que ligam os nós (d_{ij}) e, os parâmetros: α, β, ρ, Q_1

início

Inicializar as trilhas de feromônios com pequenos valores aleatórios para todos os τ_{ij}

para $w = 1$ **até** it_{max} **faça**

para $f = 1$ **até** m **faça**

Construir uma solução com base no cálculo de r_{ij}^{f} (Equação (2.12))

Atualizar os feromônios utilizando:

$\tau_{ij} = \rho\,\tau_{ij} + \Delta\tau_{ij}^{f}$ onde

$$\Delta\tau_{ij}^{f} = \begin{cases} Q_1 & \text{se a formiga } f \text{ utilizou a aresta } (i, j) \text{ na iteração } w \\ 0 & \text{caso contrário} \end{cases}$$

fim

fim

fim

Saída: Sequência de nós com menor custo (menor caminho);

A seguir, no Algoritmo 7, apresenta-se o método de atualização *ant-quantity* que difere do *ant-density* somente na forma de calcular o $\Delta\tau_{ij}^{f}$. Na atualização de feromônios por *ant-quantity*, usa-se um peso inversamente proporcional ao comprimento das arestas, reforçando o uso de arestas menores, o que contribui para encontrar o menor caminho ou, caminho de menor custo.

Algoritmo 7: *Ant System* (AS) com atualização de feromônios *ant-quantity*, adaptado de Dorigo, Maniezzo e Colorni (1991).

Entrada: número máximo de iterações (it_{max}), número de formigas (m), número de nós que cada formiga deve percorrer para completar uma solução, comprimento das arestas que ligam os nós (d_{ij}) e, os parâmetros: α, β, ρ, Q_2

início

Inicializar as trilhas de feromônios com pequenos valores aleatórios para todos os τ_{ij}

para $w = 1$ **até** it_{max} **faça**

para $f = 1$ **até** m **faça**

Construir uma solução com base no cálculo de r_{ij}^{f} (Equação (2.12))

Atualizar os feromônios utilizando:

$\tau_{ij} = \rho\ \tau_{ij} + \Delta\tau_{ij}^{f}$ onde

$$\Delta\tau_{ij}^{f} = \begin{cases} \dfrac{Q_2}{d_{ij}} & \text{se a formiga } f \text{ utilizou a aresta } (i, j) \text{ na iteração } w \\ 0 & \text{caso contrário} \end{cases}$$

fim

fim

fim

Saída: Sequência de nós com menor custo (menor caminho);

A atualização de feromônios por *ant-cycle* é apresentada a seguir no Algoritmo 8, esta atualização difere bastante das duas citadas anteriormente pois, além de uma atualização após cada formiga construir sua solução (atualização local), é feita outra atualização após todas as formigas terem construído uma solução (atualização global). A atualização global se enquadra nas ações *daemon*, citadas como opcionais no Algoritmo 5, que é o algoritmo base da metaheurística ACO.

Dorigo, Maniezzo e Colorni (1991) acreditam que a atualização por *ant-cycle* traz melhores resultados pois, a atualização após cada formiga construir sua solução leva em consideração um valor inversamente proporcional ao comprimento do caminho percorrido pela formiga (L^{f}), se o caminho for de pequeno (ou baixo custo), as arestas deste caminho receberão mais feromônios, caso contrário, se o caminho for maior (ou alto custo) as arestas receberão menos feromônios.

Algoritmo 8: *Ant System* (AS) com atualização de feromônios *ant-cycle*, adaptado de Dorigo, Maniezzo e Colorni (1991).

Entrada: número máximo de iterações (it_{max}), número de formigas (m), número de nós que cada formiga deve percorrer para completar uma solução, comprimento das arestas que ligam os nós (d_{ij}) e, os parâmetros: $\alpha, \beta, \rho_1, \rho_2, Q_3$

início

Inicializar as trilhas de feromônios com pequenos valores aleatórios para todos os τ_{ij}

para $w = 1$ **até** it_{max} **faça**

para $f = 1$ **até** m **faça**

Construir uma solução com base no cálculo de r_{ij}^f (Equação (2.12))

Calcular o comprimento (ou custo) da solução (L^f)

Atualizar os feromônios utilizando:

$$\tau_{ij} = \rho_1 \, \tau_{ij} + \Delta\tau_{ij}^f$$

onde

$$\Delta\tau_{ij}^f = \begin{cases} \dfrac{Q_3}{L^f} & \text{se a formiga } f \text{ utilizou a aresta } (i, j) \text{ na iteração } w \\ 0 & \text{caso contrário} \end{cases}$$

fim

Atualizar os feromônios utilizando:

$$\tau_{ij} = \rho_2 \, \tau_{ij} + \Delta\tau_{ij} \quad \text{onde} \quad \Delta\tau_{ij} = \sum_{f=1}^{m} \Delta\tau_{ij}^f$$

fim

fim

Saída: Sequência de nós com menor custo (menor caminho);

O algoritmo *Ant System* (AS), apresentado nesta seção, foi de grande importância para o desenvolvimento da metaheurística ACO, principalmente, por ter sido o primeiro e, a partir dele, derivaram-se outros e muitas aplicações. A seguir, na Subseção 2.5.3.2 apresenta-se o *Ant Colony System* (ACS) que foi criado a partir do AS.

2.5.3.2 Ant Colony System (ACS)

Dorigo e Gambardella (1997) propuseram um sistema colônia de formigas, *Ant Colony System* (ACS), que trouxe algumas mudanças em relação ao *Ant System* (AS), segundo os autores ACS difere-se de AS principalmente em três aspectos:

(i) para a escolha da sequência utilizada no caminho, apresenta-se uma regra de transição que, fornece uma maneira direta de equilíbrio entre a exploração de novas arestas e, exploração a *priori* de conhecimento acumulado sobre o problema;

(ii) a atualização global para os feromônios, é aplicada apenas para as arestas que pertencem a melhor solução já encontrada;
(iii) enquanto as formigas constroem uma solução, é aplicado uma atualização local para os feromônios;
No Algoritmo 9 apresenta-se o algoritmo proposto pelos autores do ACS.

Algoritmo 9: *Ant Colony System* (ACS), adaptado de Dorigo e Gambardella (1997) .

Entrada: número máximo de iterações (it_{max}), número de formigas (m), número de nós que cada formiga deve percorrer para completar uma solução (c), comprimento das arestas que ligam os nós (d_{ij}) e, os parâmetros: β, q_0.

início
  para $w = 1$ **até** it_{max} **faça**
    Posicionar cada formiga no nó inicial
    para $f = 1$ **até** m **faça**
      para $x = 1$ **até** c **faça**
        Sorteia um número aleatório para q em $[0, 1]$
        Suponha que a formiga está no nó i e vai escolher um nó j para mover-se
        se $q \leq q_0$ **então**
          Calcule o valor de $r1_{ij}$ para todos os $j \in \Omega_f$

$$r1_{ij} = \tau_{ij}\, \eta_{ij}^{\beta} \quad \text{onde} \quad \eta_{ij} = 1/d_{ij} \tag{2.13}$$

          O nó j que produzir o maior valor para $r1_{ij}$ será anexado a solução
          Aplicar a regra de atualização local na aresta (i, j)
        fim
        senão
          Calcule o valor de $r2_{ij}$ para todos os $j \in \Omega_f$

$$r2_{ij} = \frac{\tau_{ij}\, \eta_{ij}^{\beta}}{\sum_{l \in \Omega_f} \tau_{il}\, \eta_{il}^{\beta}} \tag{2.14}$$

          O nó j que produzir o maior valor para $r2_{ij}$ será anexado a solução
          Aplicar a regra de atualização local na aresta (i, j)
        fim
      fim
    fim
    Aplicar a regra de atualização global
  fim
fim

Saída: Sequência de nós com menor custo (menor caminho);

No ACS m formigas são inicialmente posicionadas em n nós (ou cidades), escolhidas de acordo com uma regra de inicialização (por exemplo, aleatoriamente). Cada formiga constrói uma solução (ou rota), aplicando a regra de transição que é, semelhante a regra do AS, são

levadas em consideração informações heurísticas e dos feromônios. Enquanto constrói a solução, a formiga modifica as quantidades de feromônios nas arestas visitadas, aplicando a regra de atualização local. Depois que todas as formigas construíram suas soluções, os feromônios são novamente atualizados com a regra de atualização global.

Na transição de um nó para outro utilizam-se duas regras. Para escolher a regra que será usada define-se um parâmetro q_0 ($0 \leq q_0 \leq 1$) e sorteia-se um número aleatório q uniformemente distribuído em $[0, 1]$, se $q \leq q_0$ utiliza-se a regra apresentada na Equação (2.13), caso contrário, utiliza-se a regra da Equação (2.14).

Este algoritmo introduz uma regra de atualização local dos feromônios, todas as formigas, após cada passo da construção da solução, atualizam o valor do feromônio da aresta que visitou de acordo com Equação (2.15), onde ρ_1 ($0 \leq \rho_1 \leq 1$) é a taxa de evaporação do feromônio e τ_0 é o valor inicial do feromônio. Os autores sugerem definir τ_0 como um valor pequeno, caso tenha-se uma solução (exata ou aproximada) para problema, por um outro método, pode-se usar o inverso deste valor.

$$\tau_{ij} \longleftarrow (1 - \rho_1)\, \tau_{ij} + \rho_1\, \tau_0 \tag{2.15}$$

A principal contribuição da regra de atualização local é, alterar dinamicamente o valor dos feromônios nas arestas, de forma que, arestas já visitas não se tornem tão atraentes para diversificação de caminhos.

Ao final de cada iteração, ou seja, após todas as formigas terem construído uma solução, é aplicada uma regra de atualização de feromônios para o caminho percorrido pela formiga que encontrou a melhor solução até a presente iteração. Esta atualização é apresentada na Equação (2.16) onde, ρ_2 é a taxa de evaporação do feromônio e, $\Delta\tau_{ij}$ deve ser escolhido de acordo com o problema, para o caso do problema do caixeiro viajante pode-se considerar $\Delta\tau_{ij} = \frac{1}{L_{best}}$, onde L_{best} é comprimento da melhor rota, até o presente momento.

$$\tau_{ij} \longleftarrow \begin{cases} (1 - \rho_2)\, \tau_{ij} + \rho_2\, \Delta\tau_{ij} & \text{se } ij \text{ pertence à solução da melhor formiga} \\ \tau_{ij} & \text{caso contrário} \end{cases} \tag{2.16}$$

A ideia desta regra é que as arestas pertencentes ao melhor caminho já construído, dentre todos os caminhos desde o inicio do algoritmo, recebam um acréscimo de feromônios. Essa atualização faz com que o espaço de busca seja direcionado para as arestas que produziram a melhor solução. Dorigo e Gambardella (1997) sugerem que a escolha dos valores dos parâmetros deve ser feita após experimentos preliminares, de acordo com o problema que será resolvido. Para resolver o problema do caixeiro viajante com n cidades eles utilizaram no máximo n formigas, $\beta = 2$, $q_0 = 0,9$ e $\rho_1 = \rho_2 = 0,1$. A seguir, na Subseção 2.5.3.3 apresenta-se o Max-Min *Ant System* (MMAS) que também traz propostas de mudanças em relação ao *Ant System* (AS).

2.5.3.3 Max-Min Ant System (MMAS)

Proposto por Stützle e Hoos (2000), esse algoritmo também apresenta algumas melhorias em relação ao *Ant System* (AS). Os autores ressaltam que a chave para encontrar um melhor desempenho em algoritmos ACO é, combinar uma exploração das melhores soluções com um mecanismo para evitar a estagnação precoce, assim, destacam três pontos importantes, apresentados a seguir, que fazem seu algoritmo se diferenciar do AS.

(i) Afim de explorar as melhores soluções encontradas, propõem-se que após cada iteração, apenas uma única formiga contribua para a atualização dos feromônios, esta formiga pode ser a que obteve a melhor solução da iteração atual ou, a que obteve a melhor solução desde o inicio do algoritmo. A atualização de feromônios é apresentada na Equação (2.17) onde ρ é a taxa de evaporação, $\Delta\tau_{ij}^{best}$ é definido na Equação (2.18), sendo que L_{best} pode ser o valor da melhor solução encontrada na iteração atual ou, da melhor solução encontrada desde o inicio do algoritmo.

$$\tau_{ij} \longleftarrow \rho\, \tau_{ij} + \Delta\tau_{ij}^{best} \tag{2.17}$$

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L_{best}} & \text{se } ij \text{ pertence a solução da melhor formiga} \\ 0 & \text{caso contrário} \end{cases} \tag{2.18}$$

(ii) Para evitar que algumas arestas tenham valores de feromônios muito discrepantes, direcionando o espaço de busca de soluções e, levando a uma convergência prematura ou a um ótimo local, limita-se os valores das trilhas de feromônios em um intervalo $[\tau_{min}, \tau_{max}]$.

O valor de τ_{max} é calculado conforme Equação (2.19) onde, ρ é a taxa de evaporação e, L_{best} é o valor da melhor solução encontrada desde o inicio do algoritmo. Assim, o valor de τ_{max} é atualizado sempre que o algoritmo encontrar uma nova melhor solução (L_{best}).

$$\tau_{max} = \frac{1}{1-\rho}\frac{1}{L_{best}} \tag{2.19}$$

Segundo Dorigo e Socha (2006) o valor para τ_{min} deve ser escolhido com bastante cautela pois, tem bastante influencia na performance do algoritmo. O valor de τ_{min} precisa ser positivo e diferente de zero, porém, pode ser definido como um valor pequeno próximo de zero ou, proporcional a um valor de τ_{max}, o ideal é fazer experimentos acerca do problema que será resolvido para decidir como defini-lo.

Stützle e Hoos (2000) apresentam uma abordagem analítica para o cálculo de τ_{min}, baseados na probabilidade p_{best}, que a formiga construirá a melhor solução encontrada até o momento. Supõe-se que a probabilidade de uma formiga tomar a decisão certa (isto é, a decisão que pertence à sequência de decisões, que conduzem à construção da melhor solução encontrada até agora) em cada um dos n passos, é dada por uma constante p_{dec}. Uma formiga tem que

tomar a decisão certa n vezes, para construir a melhor solução com probabilidade p_{dec}^{n}. Defini-se $p_{dec}^{n} = p_{best}$ assim, $p_{dec} = \sqrt[n]{p_{best}}$. Em média, em cada ponto, a formiga tem que escolher entre $avg = n/2$ opções de movimentos disponíveis. A fórmula analítica sugerida para calcular τ_{min} é apresentada na Equação (2.20).

$$\tau_{min} = \frac{\tau_{max}\,(1 - p_{dec})}{(avg - 1)\,p_{dec}} = \frac{\tau_{max}\,(1 - \sqrt[n]{p_{best}})}{(avg - 1)\,\sqrt[n]{p_{best}}} \tag{2.20}$$

A forma analítica para calcular um valor de τ_{min}, exige que o usuário defina um valor para a constante p_{best}, para isto, precisa-se fazer um estudo sobre o problema que será resolvido, Stützle e Hoos (2000) fazem alguns experimentos para relacionar os valores de p_{best} com a convergência do algoritmo e, também mostram experimentalmente, a importância do uso de limites inferiores.

(iii) Para uma maior exploração de soluções no inicio do algoritmo recomenda-se inicializar todas as trilhas de feromônios (todos os τ_{ij}) com o valor de τ_{max} . Porém, de acordo com o calculo sugerido para τ_{max}, apresentado na Equação (2.19), precisará atribuir um valor para L_{best}. A sugestão é, se possível, utilizar uma boa solução já conhecida para o problema. A ideia de inicializar as trilhas de feromônios com τ_{max}, ao invés de τ_{min}, surgiu a partir de experimentos feitos pelos autores do método com várias instâncias de problemas e, onde a inicialização com τ_{max} mostrou resultados bem superiores. A seguir, no Algoritmo 10 apresenta-se o MMAS.

Algoritmo 10: Max-Min *Ant System* (MMAS), adaptado de Stützle e Hoos (2000).

Entrada: número máximo de iterações (it_{max}), número de formigas (m), número de nós que cada formiga deve percorrer para completar uma solução n, comprimento das arestas que ligam os nós (d_{ij}) e, os parâmetros: L_{best}, ρ, p_{dec}, α e β

início

- Calcular τ_{max} utilizando uma solução L_{best} já conhecida para o problema
- Calcular τ_{min}
- Inicializar as trilhas de feromônios com $\tau_{ij} = \tau_{max} \;\; \forall\, (i, j)$
- **para** $w = 1$ **até** it_{max} **faça**
  - **para** $f = 1$ **até** m **faça**
    - Construir uma solução com base no cálculo de r_{ij}^{f} (Equação (2.12))
    - Caso a solução encontrada for melhor que a L_{best} atualizar τ_{min} e τ_{max}
  - **fim**
  - Atualizar os feromônios utilizando Equação (2.17)
  - Redefinir os feromônios no intervalo $[\tau_{min}, \tau_{max}]$ conforme Equação (2.21)
- **fim**

fim

Saída: Sequência de nós com menor custo (menor caminho);

Após cada iteração, todos os τ_{ij} são atualizados para $\overline{\tau_{ij}}$ conforme Equação (2.21) para que os valores dos feromônios permaneçam no intervalo $[\tau_{min}, \tau_{max}]$.

$$\overline{\tau_{ij}} = \begin{cases} \tau_{min} & \text{se } \tau_{ij} < \tau_{min} \\ \tau_{max} & \text{se } \tau_{ij} > \tau_{max} \\ \tau_{ij} & \text{caso contrário} \end{cases} \tag{2.21}$$

A ideia do algoritmo MMAS é, construir soluções utilizando-se da regra de transição proposta no método AS. Porém, a inicialização das trilhas de feromônios será com o valor correspondente a τ_{max} e, durante as iterações do algoritmo os valores dos feromônios se mantem no intervalo $[\tau_{min}, \tau_{max}]$. Além de um intervalo para limitação dos valores dos feromônios, outra importante contribuição, é a atualização dos valores destes feromônios, somente após todas as formigas terem construído suas soluções e, somente nas arestas pertencentes a melhor solução.

Os métodos de solução, da metaheurística colônia de formigas, apresentados nesta Seção 2.5.3, foram utilizados para resolver o problema do caixeiro viajante mas, também podem ser usados para o problema de *flow shop* permutacional, que é o tema deste trabalho. Faz-se uma adaptação, considerando que cada cidade do problema do caixeiro viajante, representa uma tarefa. Para construir uma solução (ou caminho), cada formiga deve anexar (ou "andar"por) todas as tarefas, formando assim uma sequência de tarefas.

No problema de *flow shop* permutacional, a sequência de tarefas é a mesma em todas as máquinas. O custo de cada caminho, pode ser calculado, considerando os tempos de processamento e *setup*, para a sequência de tarefas construída pela formiga. Mais detalhes da aplicação da metaheurística colônia de formigas, no problema de *flow shop* permutacional, serão descritos no Capítulo 4.

Após apresentar o problema de *flow shop* e, alguns métodos presentes na literatura para construção de sequências de tarefas, a seguir, no Capítulo 3 apresenta-se um modelo que descreve o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, que é objeto de estudo deste trabalho.

3 Modelo (MFSP)

3.1 Introdução

A eficiência de um sistema de produção depende de vários fatores, a programação da produção é um fator de impacto no bom funcionamento do sistema. Os problemas de *flow shop* são estudados a décadas e, considerados NP-*hard* (GAREY; JOHNSON, 1979). Em um problema de *flow shop* tem-se m estágios, onde em cada estágio pode-se ter uma ou mais máquinas e, tem-se n tarefas (ou trabalhos), que devem ser processadas nas máquinas ordenadamente, isto é, cada tarefa deve ser processada primeiro no estágio 1, depois no 2 e assim sucessivamente. Segundo Baker e Trietsch (2009) e Brucker (2007) pode-se classificar os problemas de *flow shop* em:

- ***flow shop* permutacional (ou puro) (*permutation (or pure) flow shop*)** - em cada estágio têm-se apenas uma máquina e, a ordem de processamento das tarefas é a mesma em todas as máquinas. A Figura 7 apresenta o funcionamento de um sistema *flow shop* permutacional com três tarefas: T1, T2 e T3. Neste caso, se T2 $\rightarrow$ T3 $\rightarrow$ T1 é a sequência de processamento na primeira máquina, então esta mesma sequência se repete nas demais máquinas.

A primeira tarefa de uma sequência, após terminar seu processamento na máquina k já pode iniciar seu processamento na máquina $k+1$. Na ilustração apresentada na Figura 7, T2 é a primeira tarefa da sequência, quando T2 terminar seu processamento na primeira máquina, ela pode imediatamente iniciar seu processamento na segunda máquina e, assim sucessivamente, nas próximas máquinas.

As demais tarefas, após terminarem seu processamento em uma máquina k, só podem iniciar o processamento na máquina $k+1$, depois da conclusão do processamento da tarefa anterior. Na Figura 7, T1 só pode ser processada na segunda máquina, após concluir seu processamento na primeira máquina e, condicionada a liberação da mesma, sendo que a segunda máquina só será liberada, após ter processado T3.

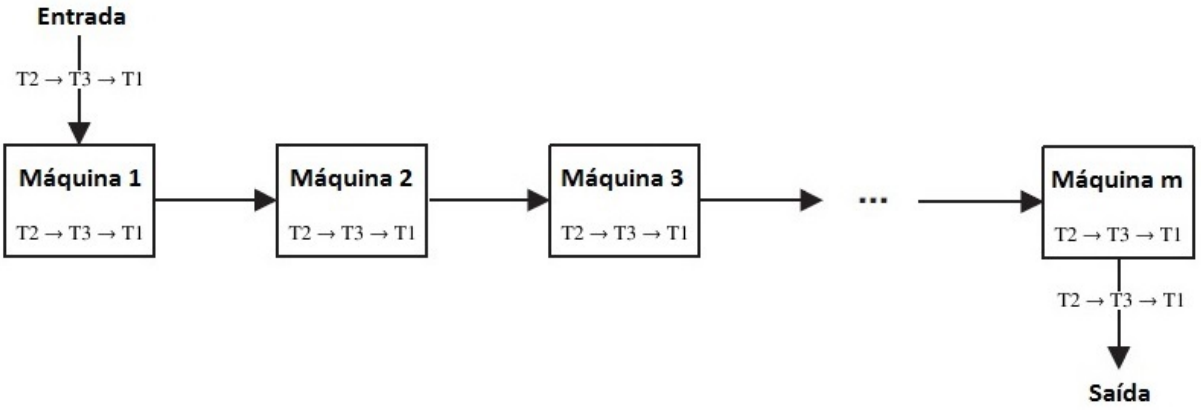

Figura 7 – Funcionamento de um sistema *flow shop* permutacional. Adaptado de: Brucker (2007)

- ***flow shop* (ou *flow shop* geral)** - em cada estágio tem-se apenas uma máquina e, todas as tarefas têm o mesmo fluxo de processamento nas máquinas, ou seja, não permite-se que uma tarefa seja processada primeiro numa máquina $k + 1$ e depois numa máquina k. Cada tarefa pode ter seu próprio caminho e, não precisa passar em todas as máquinas mas, deve respeitar o fluxo de processamento das máquinas. A sequência de processamento das tarefas não precisa ser a mesma em todas as máquinas.

A Figura 8 ilustra este funcionamento sendo que, as entradas e saídas representam que a qualquer momento pode-se receber novas tarefas (que não precisavam ser processadas nas máquinas anteriores) e também, as tarefas podem sair a qualquer momento (se não precisarem passar em outras máquinas). Observa-se que na primeira máquina a sequência de tarefas é T2 → T3 → T1, suponha que T3 não precisa passar pela máquina 2 então, a tarefa T3 pode adiantar-se e ir direto para a máquina 3.

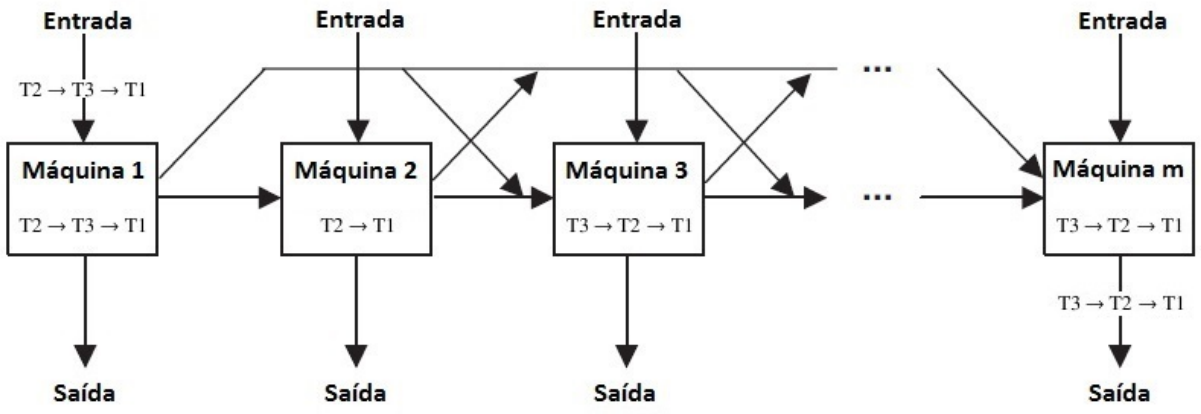

Figura 8 – Funcionamento de um sistema *flow shop* geral. Adaptado de: Brucker (2007)

- ***flow shop* híbrido (ou flexível)** - em cada estágio pode-se ter mais que uma máquina em paralelo, todas as tarefas devem passar por todos os estágios, na mesma sequência e, em cada estágio, a tarefa deve ser processada por uma única máquina.

Nestes problemas, a sequência de tarefas pode ser dependente ou independente dos tempos de preparação de máquina (tempos de *setup*). Numa sequência independente, os tempos de *setup* não são considerados ou, são incluídos nos tempos de processamento. Na prática, usa-se essa consideração quando os tempos de preparação são muito pequenos ou inexistentes.

No caso da sequência ser dependente dos tempos de *setup*, considera-se um tempo de preparação de máquina, este tempo varia de acordo com a última tarefa que foi processada na máquina e a tarefa que será processada naquele momento. Portanto, a ordem das tarefas é muito importante, pois, a soma total de todos os tempos de *setup*, influencia fortemente no tempo total de produção e, esta soma altera-se dependendo da ordem das tarefas. A seguir apresenta-se o modelo para o problema estudado neste trabalho.

3.2 Apresentação do modelo (MFSP)

O problema estudado neste trabalho, é o *flow shop* permutacional com sequência dependente dos tempos de *setup*, que é um problema de sequenciamento de n tarefas diferentes a serem processadas em m máquinas. Cada tarefa possui um tempo de processamento em cada máquina, e cada máquina, possui um tempo de *setup* que depende da tarefa que foi processada anteriormente e da que será processada em seguida. Todas as tarefas seguem a mesma sequência em todas as máquinas e, cada tarefa só pode ser processada em uma máquina por vez, como também, cada máquina só pode processar uma tarefa por vez. Não é permitido *preemption*, ou seja, não admite-se interromper o processamento das tarefas. O objetivo é minimizar o *makespan* (C_{max}), que é o instante de término da última tarefa, na última máquina.

Consideram-se tempos de *setup* assimétricos, ou seja, o tempo de preparação da máquina para processar a tarefa 1 depois a tarefa 2 é diferente do tempo de preparação oposto, primeiro a tarefa 2, depois a tarefa 1. Adota-se o *antecipatory setup*, em que a preparação de uma máquina k, para uma tarefa j, é iniciada logo após a liberação da máquina k, pela tarefa anterior. Não considera-se tempo de *setup* para a primeira tarefa a ser processada em todas as máquinas. Um modelo que contemple as restrições mencionadas é apresentado a seguir nas Equações (3.1) até (3.12).

Considere n como o número de tarefas, m o número de máquinas e os seguintes parâmetros: V é um número suficientemente grande, $s_{i,j,k}$ é o tempo de *setup* da tarefa i para tarefa j, na máquina k e $p_{j,k}$ é o tempo de processamento da tarefa j na máquina k.

As variáveis $x_{i,j,k}$ são binárias, se $x_{i,j,k} = 1$ significa que a tarefa i precede imediatamente a tarefa j na máquina k, e $x_{i,j,k} = 0$, caso contrário. As tarefas 0 e $n + 1$ representam tarefas

fictícias de início e fim de uma sequência. Se $x_{0,j,k} = 1$ significa que a tarefa j será a primeira tarefa a ser processada na máquina k, caso contrário, $x_{0,j,k} = 0$. Se $x_{i,n+1,k} = 1$ significa que a tarefa i será a última tarefa a ser processada na máquina k, caso contrário, $x_{i,n+1,k} = 0$. $C_{j,k}$ é o instante de conclusão da tarefa j na máquina k.

MODELO MFSP:

$$\text{Minimizar } C_{max} \tag{3.1}$$

Sujeito a:

$$\sum_{\substack{i=0 \\ i \neq j}}^{n} x_{i,j,k} = 1 \qquad j = 1, ..., n \qquad k = 1, ..., m \tag{3.2}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{n+1} x_{i,j,k} = 1 \qquad i = 1, ..., n \qquad k = 1, ..., m \tag{3.3}$$

$$\sum_{j=1}^{n} x_{0,j,k} = 1 \qquad k = 1, ..., m \tag{3.4}$$

$$\sum_{i=1}^{n} x_{i,n+1,k} = 1 \qquad k = 1, ..., m \tag{3.5}$$

$$\begin{aligned} &x_{i,j,1} = ... = x_{i,j,m} && i, j = 1, ..., n \quad \forall\, i \neq j \\ &x_{0,j,1} = ... = x_{0,j,m} && j = 1, ..., n \\ &x_{i,n+1,1} = ... = x_{i,n+1,m} && i = 1, ..., n \end{aligned} \tag{3.6}$$

$$C_{j,k} \geq C_{i,k} + x_{i,j,k}\,(s_{i,j,k} + p_{j,k}) + V\,(x_{i,j,k} - 1) \quad i = 0, ..., n \quad j = 1, ..., n \quad \forall\, i \neq j \quad k = 1, ..., m \tag{3.7}$$

$$C_{j,k+1} \geq C_{j,k} + p_{j,k+1} \qquad j = 1, ..., n \qquad k = 1, ..., m - 1 \tag{3.8}$$

$$C_{max} \geq C_{j,m} \qquad j = 1, ..., n \tag{3.9}$$

$$C_{0,1} = 0 \tag{3.10}$$

$$C_{0,k} \geq C_{0,k-1} + \sum_{j=1}^{n}(x_{0,j,k-1}\ p_{j,k-1}) \qquad k = 2, ..., m \tag{3.11}$$

$$\begin{aligned} &x_{i,j,k} \in \{0,1\} \quad i = 0, ..., n \quad j = 1, ..., n+1 \quad \forall\, i \neq j \quad k = 1, ..., m \\ &C_{j,k} \in \mathbb{R}_{+} \qquad j = 0, ..., n \quad k = 1, ..., m \end{aligned} \tag{3.12}$$

A função objetivo, minimizar o *makespan* é apresentada na Equação (3.1). As restrições da Equação (3.2) asseguram que cada tarefa j só pode ser precedida por outra única tarefa e, as restrições da Equação (3.3) asseguram que cada tarefa i só pode ter uma única tarefa sucessora. A garantia que somente uma tarefa será a inicial em cada máquina k está na Equação (3.4) e, assegura-se que a última tarefa será única em todas as k máquina na Equação (3.5). Na Equação (3.6) afirma-se que a sequência de tarefas deve ser a mesma em todas as máquinas.

As restrições da Equação (3.7) são usadas para calcular os tempos de conclusão de todas as tarefas. Definiu-se que C_{ik} representa o instante de término do processamento da tarefa i na máquina k e, consequentemente, C_{jk} representa o instante de término do processamento da tarefa j nesta mesma máquina k. Se a tarefa i precede a tarefa j, então a restrição $C_{ik} < C_{jk}$ não é suficiente. Precisa-se limitar inferiormente o tempo de conclusão da tarefa j, na máquina k, com o tempo de conclusão do processamento da tarefa anterior i, acrescido do tempo *setup* entre estas tarefas e, o tempo de processamento desta tarefa j na máquina k. Caso a tarefa i não preceda imediatamente a tarefa j, tem-se $x_{ijk} = 0$, fazendo com que a restrição limite simplesmente que C_{jk} é maior que um número negativo, tendo em vista que V é definido como um número suficientemente grande.

A Figura 9, ilustra as restrições da Equação (3.7). Têm-se um sequenciamento de tarefas T3 → T4 → T2 → T1, na primeira máquina (M1). Observa-se que a tarefa T4 precede imediatamente a tarefa T2, neste caso $x_{421} = 1$ assim:

$$\begin{aligned} &C_{21} \geq C_{41} + x_{421}\ (s_{421} + p_{21}) + V\ (x_{421} - 1) && => \\ &C_{21} \geq C_{41} + 1\ (s_{421} + p_{21}) + V\ (1 - 1) && => \\ &C_{21} \geq C_{41} + \ (s_{421} + p_{21}) \end{aligned}$$

Observa-se que a tarefa T4 não precede imediatamente a tarefa T1, neste caso $x_{411} = 0$, considera-se que V é um valor suficientemente grande, têm-se:

$$\begin{aligned} &C_{11} \geq C_{41} + x_{411}\ (s_{411} + p_{11}) + V\ (x_{411} - 1) && => \\ &C_{11} \geq C_{41} + 0\ (s_{411} + p_{11}) + V\ (0 - 1) && => \\ &C_{11} \geq C_{41} - V && => \\ &C_{11} \geq -V \end{aligned}$$

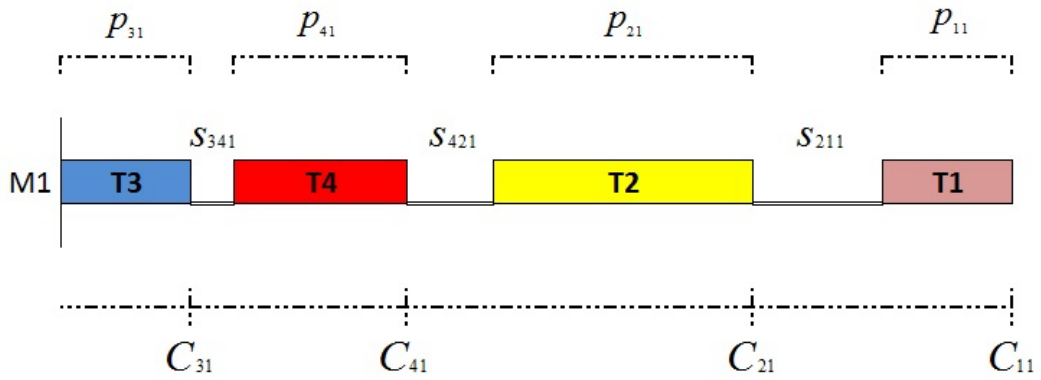

Figura 9 – Ilustração das restrições da Equação (3.7). Fonte: o autor.

As restrições da Equação (3.8) asseguram que o tempo de conclusão de uma tarefa j na máquina $k+1$ ($C_{j,k+1}$) tem que ser maior que o tempo de conclusão desta tarefa na máquina anterior k ($C_{j,k}$) acrescido do tempo de processamento da tarefa j na máquina $k+1$ ($p_{j,k+1}$).

A Figura 10 ilustra as restrições da Equação (3.8). Observa-se, por exemplo, que a o tempo de conclusão do processamento da tarefa T2, na segunda máquina M2 (C_{22}), é maior que o tempo de conclusão desta mesma tarefa T2, na primeira máquina M1 (C_{21}), acrescido do tempo de processamento da tarefa T2 na segunda máquina (p_{22}), ou seja, $C_{22} \geq C_{21} + p_{22}$. É possível observar que esta restrição se mantém para todas as tarefas.

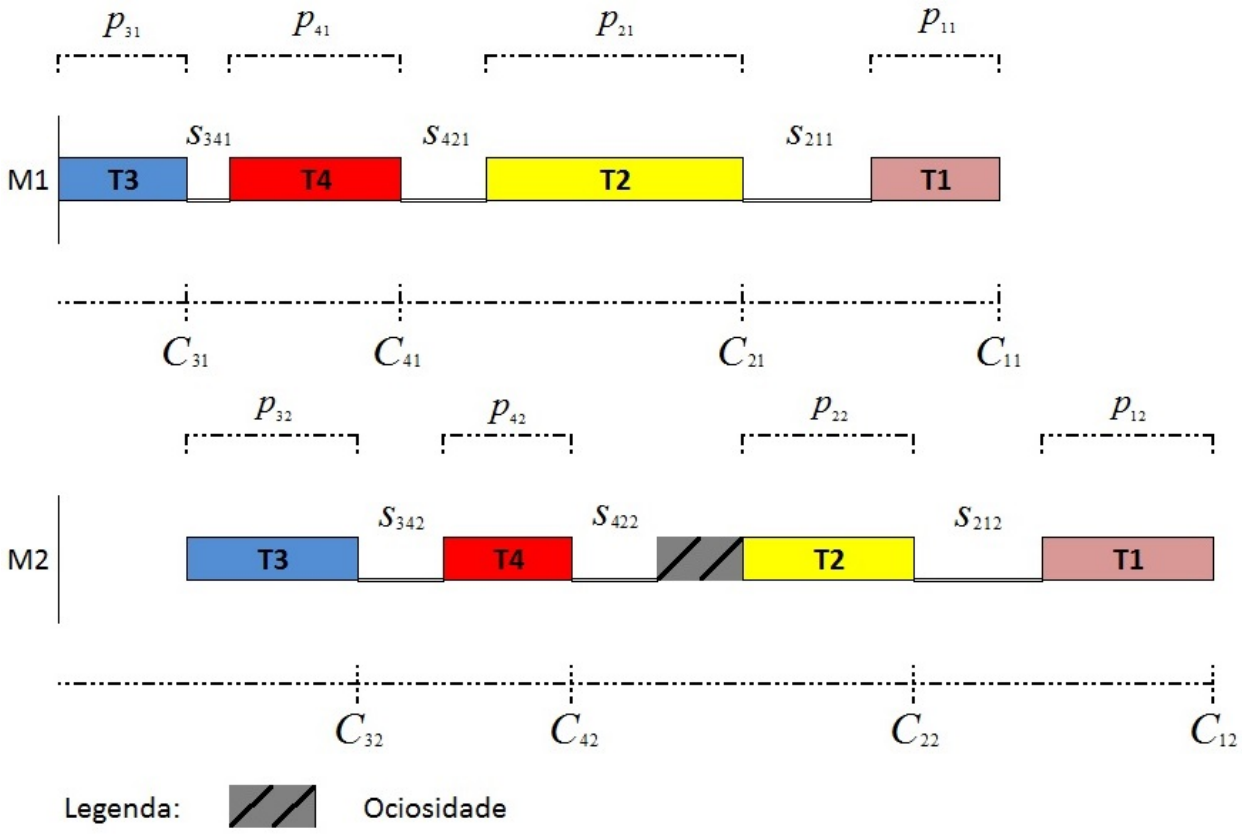

Figura 10 – Ilustração das restrições da Equação (3.8). Fonte: o autor.

Na Equação (3.9) define-se que a variável C_{max}, deve ser maior que o tempo de conclusão do processamento, de qualquer tarefa j, na última máquina m (C_{jm}). Sabe-se que o maior tempo de conclusão de processamento, na última máquina, será da última tarefa. Considere um problema aleatório de quatro tarefas (T1, T2, T3 e T4) e, duas máquinas (M1 e M2), conforme ilustração apresentada na Figura 10. Neste caso, representa-se o tempo de conclusão da última tarefa da sequência (T1), na última máquina (M2), por C_{12} portanto, $C_{max} \geq C_{12}$.

Criou-se tarefas fictícias 0 e $n+1$, para delimitar o início e o fim da sequência de tarefas. Com auxílio destas tarefas, as restrições apresentadas nas Equações (3.4) e (3.5) forçam que a tarefa inicial da sequência e, a tarefa final, sejam únicas. A criação destas duas tarefas fictícias não afeta o cálculo da função objetivo pois, não são considerados tempos de processamento para elas.

A tarefa fictícia 0 é importante para delimitar, a partir de que momento, a primeira tarefa é processada na próxima máquina. Na ilustração da Figura 10, observa-se que a primeira tarefa inicia seu processamento no instante zero, isto é definido no modelo pela Equação (3.10), onde fixa-se que o tempo de conclusão da tarefa fictícia 0, na primeira máquina é zero.

A partir da segunda máquina $k = 2, 3, ..., m$, a primeira tarefa da sequência, inicia seu processamento imediatamente, após ter concluído o processamento na máquina anterior.

No modelo, essa delimitação é imposta na Equação (3.11), utilizando-se a tarefa fictícia 0. Considera-se que o tempo de conclusão da tarefa fictícia 0, limita o tempo de início da primeira tarefa, em cada máquina. Esse limite inferior, é composto pela soma do tempo de conclusão do processamento da tarefa fictícia, na máquina anterior, com o tempo de processamento da primeira tarefa, também na máquina anterior. Uma ilustração para estas restrições é apresentada na Figura 11.

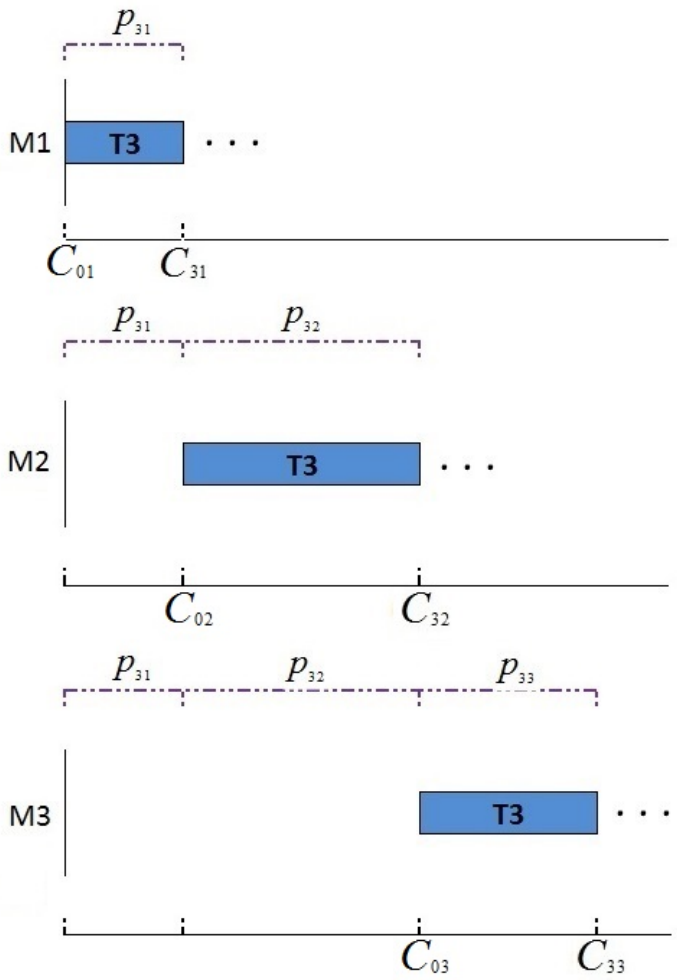

Figura 11 – Ilustração das restrições da Equação (3.11). Fonte: o autor.

Na ilustração apresentada na Figura 11, evidencia-se que a primeira tarefa da sequência (T3), só inicia seu processamento na segunda máquina (M2), após ter concluído o processamento na primeira máquina. O tempo de início do processamento da primeira tarefa T3, é limitado pelo tempo de conclusão do processamento da tarefa fictícia 0. O tempo de conclusão da tarefa fictícia 0, na máquina k, é limitado inferiormente pelo tempo de conclusão desta tarefa na máquina anterior $k-1$, somado ao tempo de processamento da primeira tarefa. Considera-se este exemplo aleatório em que, a tarefa T3 é a primeira tarefa da sequência então, $x_{03k} = 1$ para todas as k máquinas e, $x_{0jk} = 0$ para todo $j \neq 3$, em todas as k máquinas, assim para $k = 2$ têm-se:

$$C_{02} \geq C_{01} + x_{031}\ p_{31} \qquad => \\ C_{02} \geq C_{01} + p_{31}$$

Para $k = 3$, como T3 é a primeira tarefa da sequência, $x_{032} = 1$ e têm-se:

$$C_{03} \geq C_{02} + x_{032}\ p_{32} \qquad => \\ C_{03} \geq C_{02} + p_{32}$$

Na Equação (3.12) define-se os tipos de variáveis.

O modelo apresentado neste capítulo, foi aplicado para várias instâncias de problemas, geradas aleatoriamente e resolvidas com o otimizador CPLEX *Interactive Optimizer* 12.4.0.0. Os resultados serão apresentados no Capítulo 5.

A seguir, no Capítulo 4, apresenta-se um algoritmo baseado na metaheurística colônia de formigas que, inicializa as trilhas artificiais de feromônios a partir de soluções iniciais para que, formigas artificiais construam soluções para o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, apresentado no modelo acima (MFSP), nas Equações (3.1) até (3.12).

4 Descrição do algoritmo proposto (AFSP)

Neste capítulo propõe-se um algoritmo, baseado em ACO, para resolver o problema de *flow shop* permutacional, apresentado no modelo MFSP, no Capítulo 3. O algoritmo proposto envolve a criação de sequências iniciais, através de heurísticas construtivas e, a partir destas sequências, inicializa-se as trilhas de feromônios, para aplicar a metaheurística colônia de formigas, para resolver o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*.

A aplicação de um metaheurística é necessária para resolver este tipo de problema pois, a resolução por uma metodologia exata, como o modelo MFSP, dificilmente resolve problemas de médio e grande porte e, quando resolve precisa de um tempo computacional muito grande. Com uma metaheurística consegue-se chegar a uma boa solução, em alguns casos chega-se na melhor solução porém, o tempo gasto para chegar a esta aproximação tende a ser muito menor do que o tempo gasto com uma metodologia exata.

Conforme levantamento feito na Seção 2.2, Ying e Liao (2004), Yagmahan e Yenisey (2008), Yagmahan e Yenisey (2010) e Ahmadizar (2012) resolveram o problema de *flow shop* permutacional, com algoritmos baseados na metaheurística colônia de formigas, sem considerar a sequência dependente dos tempos de *setup*. Gajpal, Rajendran e Ziegler (2006) e Mirabi (2011) também resolveram o problema de *flow shop* permutacional, com algoritmos baseados na metaheurística colônia de formigas mas, considerando a sequência dependente dos tempos de *setup*. Em relação ao que já foi publicado, este trabalho faz uma consideração diferente de tempos de *setup* nas regras de transição, que são utilizadas para construir a sequência de tarefas.

Evidencia-se também, na Seção 2.2 em Li e Zhang (2012), Vanchipura e Sridharan (2013), Saravanan, Vijayakumar e Srinivasan (2014), Mirabi (2014) e Shen, Gupta e Buscher (2014) que é crescente o uso de outros algoritmos heurísticos e metaheurísticos, para resolver o problema de *flow shop* permutacional com sequência dependente dos tempos de *setup*.

Neste capítulo, na Seção 4.1, apresenta-se a estrutura geral do algoritmo proposto para este trabalho. Na Seção 4.2, exibe-se as propostas utilizadas neste trabalho, para gerar soluções iniciais, com base nos métodos presentes na literatura, já apresentados na Seção 2.4. Em seguida, nas Seções 4.3, 4.4, 4.5, 4.6, 4.7 e 4.8, expõe-se os outros procedimentos do algoritmo: processo de busca de soluções vizinhas, inicialização de feromônios, regra de transição, atualização global e local e, o critério de parada.

4.1 Estrutura geral do algoritmo proposto (AFSP)

A ideia de um algoritmo baseado em ACO (otimização colônia de formigas) é, imitar o comportamento real, das formigas que se movimentam guiadas pelas trilhas de feromônios. Problemas de otimização podem ser representados em um grafo então, projeta-se formigas artificiais, que caminham pelas arestas do grafo para encontrar soluções. Dependendo do problema que será resolvido, o objetivo pode ser encontrar o menor caminho entre dois nós do grafo, nó de origem e nó de destino. O nó de destino pode ser associado a fonte de alimento e, o nó de origem, pode ser associado ao ninho, de onde a formiga parte para buscar alimento, ou no caso de formigas artificiais, parte para construir uma solução.

Também, quando considera-se um grafo pode-se pensar em outras situações, como o problema do caixeiro viajante, que consiste em determinar a menor rota, partindo de um nó (ou cidade) de origem, para visitar todos os nós do grafo (visitando cada nó uma única vez) e, voltar ao nó de origem. No problema de *flow shop* permutacional, que é o tema de estudo deste trabalho, considera-se esta abordagem. Cada tarefa será representada por um nó do grafo, que carrega informações a respeito dos tempos de processamento, as arestas que ligam os nós, carregam informações a respeito dos tempos de *setup* entre as tarefas. Para construir uma solução (ou caminho), cada formiga deve anexar (ou "andar" por) todas as tarefas, formando assim uma sequência de tarefas.

As formigas iniciam com uma sequência vazia e, vão anexando tarefas (ou trabalhos) ao longo do processo, de modo a formar uma sequência com todas as tarefas. Para movimentar-se de uma tarefa para outra, ou seja, para escolher a próxima tarefa a ser anexada na sequência, as formigas movimentam-se de acordo com uma regra de transição, que utiliza informações das trilhas de feromônios e, informações a respeito do problema.

Dependendo do problema que está sendo resolvido, pode-se considerar diferentes informações nas regras de transição. No problema abordado neste trabalho, pode-se considerar, por exemplo, o tempo de processamento da tarefa que será processada, ou o tempo de *setup* entre a tarefa atual e a próxima que será anexada, como também, observar se nas sequências construídas anteriormente, que obtiveram um bom *makespan*, essa sequência de tarefas estava presente. Na Seção 2.5.3 apresentou-se os principais algoritmos, presentes na literatura, baseados na otimização colônia de formigas (ACO).

A estrutura geral do algoritmo proposto nesse trabalho está representada em Algoritmo 11. Inicialmente, gera-se sequências iniciais por métodos que serão explicitados na Seção 4.2. Para melhorar estas sequências iniciais, aplica-se um procedimento de busca de soluções vizinhas, Seção 4.3. Após, inicializa-se as trilhas de feromônios, conforme descrito na Seção 4.4. As formigas constroem seus caminhos segundo a regra de transição apresentada na Seção 4.5.

Ao final da rota de cada formiga, os feromônios são atualizados de acordo com uma regra de atualização local, exposta na Seção 4.6. Depois que todas as formigas completaram suas

rotas, os feromônios são atualizados de acordo com uma regra de atualização global, descrita na Seção 4.7, sendo que a melhor rota até o momento, a que obteve melhor *makespan*, recebe um acréscimo de feromônios.

Algoritmo 11: Estrutura geral do algoritmo proposto (AFSP).

Entrada: Conjunto de parâmetros e conjunto de dados: número de tarefas e máquinas, matrizes com os tempos de processamento e *setup*;

início

- Gerar sequências iniciais;
- Busca de soluções vizinhas;
- Inicializar as trilhas de feromônios;
- **enquanto** *não atender as condições de parada* **faça**
  - **para** $f = 1$ **até** *número de formigas* **faça**
    - Aplicar uma regra de transição até construir uma solução;
    - Calcular o *makespan* dessa solução;
    - Aplicar a regra de atualização local;
    - Atualizar os feromônios dentro dos limites τ_{min} e τ_{max};
  - **fim**
  - Aplicar uma regra de atualização global;
  - Atualizar os feromônios dentro dos limites τ_{min} e τ_{max};
- **fim**

fim

Saída: Melhor sequência de tarefas;

Nas próximas seções, será explicitado com mais detalhes cada parte do Algoritmo 11.

4.2 Soluções iniciais

Soluções iniciais são importantes aliadas na resolução de problemas, na metaheurística colônia de formigas é necessário inicializar as trilhas de feromônios de alguma forma, as soluções iniciais podem ser uma opção para a inicialização de feromônios.

Gajpal, Rajendran e Ziegler (2006) e Mirabi (2011) utilizaram a metaheurística colônia de formigas para um problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*. Nos dois trabalhos a inicialização dos feromônios foi feita a partir de soluções iniciais. Conforme apresentado na Subseção 2.4.4, Gajpal, Rajendran e Ziegler (2006) geraram uma sequência inicial por uma regra que chamaram relação de preferência heurística e utilizaram o valor do *makespan* desta sequência inicial para inicializar os feromônios. Mirabi (2011), inicializou os feromônios a partir de soluções iniciais, construídas com a heurística NEH modificada e um método que considera pesos para as sequências de tarefas, sendo que esses pesos dependem dos tempos de processamento das tarefas e do tempo de *setup* inicial, conforme já exposto na Subseção 2.4.5.

Neste trabalho utilizou-se 6 métodos diferentes para construção de soluções iniciais, aliados com um procedimento de busca local, estes métodos serão chamados de: PALMER1, PALMER2, NEH1, NEH2, PESO1 e PESO2 e apresentados, a seguir, nas Subseções 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.5 e 4.2.6, respectivamente. Alguns desses métodos já foram aplicados em trabalhos da literatura e foram explicitados anteriormente na Seção 2.4 e, outros, são derivados desses métodos acrescentando considerações do tempo de *setup*. A intenção é analisar quais desses métodos produzem soluções iniciais melhores e também, o quão importante é considerar os tempos de *setup* na construção das soluções. A seguir, apresenta-se o método PALMER1, baseado no trabalho de Palmer (1965).

4.2.1 PALMER1

O método PALMER1, para gerar soluções iniciais, proposto neste trabalho utiliza-se do *slope index* proposto por Palmer (1965), apresentado na Subseção 2.4.1, acrescido de um processo para gerar soluções vizinhas.

Algoritmo 12: Método PALMER1.

Entrada: Número de tarefas (n), número máquinas (m) e tempos de processamento (p_{ik} da tarefa i na máquina k);

início

  para $i = 1$ **até** n **faça**

    $$Q_i = \sum_{k=1}^{m} (2k - m - 1)p_{ik}$$

  fim

  Organizar uma sequência de tarefas em ordem decrescente dos valores de Q_i;

  Saída: Sequência de tarefas;

  Busca de soluções vizinhas;

  Saída: Melhor sequência de tarefas.

fim

No algoritmo acima, utiliza-se o cálculo do *slope index*, este índice prioriza tarefas com tempos de processamento que tendem a diminuir de máquina para máquina. Porém, não faz nenhuma consideração a respeito dos tempos de *setup*. Ao final do algoritmo é feito uma busca de soluções vizinhas e, escolhe-se a melhor sequência de tarefas encontrada dentre todas. A seguir, apresenta-se o método PALMER2, onde propõe-se uma consideração dos tempos de *setup* no cálculo do *slope index*.

4.2.2 PALMER2

PALMER2 difere-se de PALMER1 somente no cálculo do *slope index*, com o objetivo de fazer considerações a respeito dos tempos de *setup*. Quando tem-se uma sequência dependente dos tempos de *setup*, a melhor sequência de tarefas será, a que produzir os menores tempos de

setup. A ideia deste método é acrescentar um valor, referente aos tempos de*setup* no cálculo de Q_i.

O tempo médio de *setup* de uma tarefa i, em todas as máquinas, pode ser expresso por $s_{i\ medio}$, conforme Equação (4.1) onde, n é o número de tarefas, m é o número de máquinas e s_{ijk} é o tempo de *setup* da tarefa i para j na máquina k.

$$s_{i\ medio} = \frac{\sum_{k=1}^{m} \sum_{\substack{j=1, \\ j \neq i}}^{n} s_{ijk}}{n-1} \qquad \forall i \tag{4.1}$$

A proposta do método PALMER2, assim como PALMER1 é, organizar a sequência de tarefas com base no valor *slope index* Q_i, que prioriza tarefas com tempo de processamento que tendem a diminuir de máquina para máquina, ou seja, quanto maior o valor de Q_i deve-se dar preferência no sequenciamento daquela tarefa. Portanto, não pode-se acrescentar diretamente o valor de $s_{i\ medio}$ no valor de Q_i, pois deve-se acrescentar um valor maior a Q_i se os tempos de *setup* forem pequenos e, acrescentar um valor pequeno, se os tempos de *setup* forem grandes. Para isso, será usado o inverso da soma do tempo médio de *setup* de uma tarefa i, em todas as máquinas, conforme Equação (4.2).

$$\frac{1}{s_{i\ medio}} = \frac{n-1}{\sum_{k=1}^{m} \sum_{\substack{j=1, \\ j \neq i}}^{n} s_{ijk}} \qquad \forall i \tag{4.2}$$

O inverso do tempo médio de *setup* da tarefa i, em todas as máquinas, será um valor pequeno, para que o tempo de *setup* tenha impacto no valor de Q_i soma-se o inverso do tempo médio de *setup* multiplicado por uma constante δ, pré-definida, conforme apresentado na Equação (4.3).

$$Q_i = \sum_{k=1}^{m} (2k - m - 1) p_{ik} + \delta \frac{n-1}{\sum_{k=1}^{m} \sum_{\substack{j=1, \\ j \neq i}}^{n} s_{ijk}} \qquad \forall i \tag{4.3}$$

No Algoritmo 13 apresenta-se o método PALMER2 proposto. O valor de δ, pode ser escolhido de acordo com o problema que será resolvido.

Algoritmo 13: Método PALMER2.

Entrada: Número de tarefas (n), número máquinas (m), tempo de processamento (p_{ik} da tarefa i na máquina k) e tempo de *setup* (s_{ijk} da tarefa i para tarefa j na máquina k); δ;

início

para $i = 1$ **até** n **faça**

$$Q_i = \sum_{k=1}^{m} (2k - m - 1)p_{ik} + \delta \frac{n-1}{\sum_{k=1}^{m} \sum_{\substack{j=1, \\ j \neq i}}^{n} s_{ijk}}$$

fim

Organizar uma sequência de tarefas em ordem decrescente dos valores de Q_i;

Saída: Sequência de tarefas;

Busca de soluções vizinhas;

Saída: Melhor sequência de tarefas.

fim

Para os problemas resolvidos neste trabalho, em que os resultados serão apresentados no Capítulo 5, definiu-se δ em função do número de máquinas então, quanto maior o número de máquinas, maior será o valor de δ. Após a busca de soluções vizinhas, no final do algoritmo, a saída será a melhor sequência de tarefas, dentre todas as sequências avaliadas. A seguir, apresenta-se o método NEH1.

4.2.3 NEH1

O método NEH1, proposto neste trabalho para gerar soluções iniciais, foi baseado na heurística NEH e NEH modificada apresentadas nas Subseções 2.4.2 e 2.4.3, respectivamente. Os dois primeiros passos são os mesmos da heurística NEH. O Algoritmo deste método é apresentado no Algoritmo 14.

Neste algoritmo, o fato de calcular o índice T_i para cada tarefa i e, organizá-las em uma sequência de acordo com a ordem decrescente destes valores, vêm da heurística NEH (NAWAZ; ENSCORE; HAM, 1983). Já, a busca por soluções vizinhas, é inspirada no que foi proposto na heurística NEH modificada (RUIZ; MAROTO; ALCARAZ, 2005) que, escolhe duas tarefas aleatórias e troca com as duas primeiras tarefas da sequência. O processo de busca de soluções vizinhas utilizado neste trabalho, não é o mesmo, e será exposto com mais detalhes na Seção 4.3. Após a busca de soluções vizinhas, no final do algoritmo, a saída será a melhor sequência de tarefas, dentre todas as sequências.

Algoritmo 14: Método NEH1.

Entrada: Número de tarefas (n), número máquinas (m) e tempo de processamento (p_{ik} da tarefa i na máquina k);

início

para $i = 1$ **até** n **faça**

$$T_i = \sum_{k=1}^{m} p_{ik}$$

fim

Organizar uma sequência de tarefas em ordem decrescente dos valores de T_i;

Saída: Sequência de tarefas;

Busca de soluções vizinhas;

Saída: Melhor sequência de tarefas.

fim

A seguir, na Subseção 4.2.4, apresenta-se um método baseado na heurística NEH e NEH modificada, acrescentando considerações em relação aos tempos de *setup*.

4.2.4 NEH2

Em NEH1, após calcular os índices T_i, organiza-se a sequência em ordem decrescente em relação aos valores destes, ou seja, seleciona-se para o início da sequência, as tarefas que têm a maior soma dos tempos de processamento em todas as máquinas. Quando tem-se uma sequência dependente dos tempos de *setup*, o ideal é que a sequência de tarefas tenha um tempo de *setup* pequeno, portanto considerar os tempos de *setup* no cálculo de T_i pode trazer boas soluções.

A proposta deste trabalho para o método NEH2, difere-se do NEH1 somente no cálculo de T_i, onde além da soma dos tempos de processamento, soma-se uma parcela que faz considerações a respeito dos tempos de *setup* daquela tarefa, conforme o que foi proposto para o método PALMER2, na Subseção 4.2.2.

O tempo médio de *setup* de uma tarefa i, em todas as máquinas, pode ser expresso por $s_{i\ medio}$, conforme Equação (4.1) e, o inverso dele é exposto na Equação (4.2). Como em PALMER2, a ideia deste método é, acrescentar um valor referente aos tempos de *setup* no cálculo de T_i. O inverso do tempo médio de *setup* da tarefa i, em todas as máquinas, será um valor pequeno, para que o tempo de *setup* tenha impacto no valor de T_i soma-se o inverso do tempo médio de *setup*, multiplicado por uma constante δ pré-definida, conforme apresentado na Equação (4.4).

$$T_i = \sum_{k=1}^{m} p_{ik} + \delta \frac{n-1}{\sum_{k=1}^{m} \sum_{\substack{j=1, \\ j \neq i}}^{n} s_{ijk}} \qquad \forall i \tag{4.4}$$

No Algoritmo 15 apresenta-se o método NEH2 proposto.

Algoritmo 15: Método NEH2.

Entrada: Número de tarefas (n), número máquinas (m), tempo de processamento (p_{ik} da tarefa i na máquina k) e tempo de *setup* (s_{ijk} da tarefa i para tarefa j na máquina k); δ;

início

$\delta = 100m$

para $i = 1$ **até** n **faça**

$$T_i = \sum_{k=1}^{m} p_{ik} + \delta \frac{n-1}{\sum_{k=1}^{m} \sum_{\substack{j=1, \\ j \neq i}}^{n} s_{ijk}}$$

fim

Organizar uma sequência de tarefas em ordem decrescente dos valores de T_i;

Saída: Sequência de tarefas;

Busca de soluções vizinhas;

Saída: Melhor sequência de tarefas.

fim

A constante δ deve ser definida de acordo com o problema que está sendo resolvido. Neste trabalho, definiu-se δ em função do número de máquinas então, quanto maior o número de máquinas, maior será o valor de δ. No final do algoritmo, após a busca de soluções vizinhas, a saída será a melhor sequência de tarefas, dentre todas as sequências.

A seguir, apresenta-se o método PESO1, baseado no que propôs Mirabi (2011) que resolveu um problema de *flow shop* permutacional, considerando a sequência dependente dos tempos de *setup*, com um algoritmo baseado na metaheurística colônia de formigas.

4.2.5 PESO1

Na Subseção 2.4.5, especificamente no Algoritmo 3, apresenta-se um método proposto por Mirabi (2011) quando resolveu um problema de *flow shop* permutacional, considerando a sequência dependente dos tempos de *setup*, com um algoritmo baseado na metaheurística colônia de formigas.

Neste método, Mirabi (2011) propôs calcular um peso z_i para cada tarefa i depois, calcula-se o peso de cada aresta (i, j) fazendo $w_{ij} = z_j - z_i$. Segundo o que foi proposto por ele, este peso z_i carrega informações a respeito do tempo médio de processamento da tarefa i em todas as m

máquinas e a soma dos tempos de *setup* inicial, que é, a soma dos tempos de *setup* para processar uma tarefa i como primeira tarefa em todas as máquinas ($\sum_{k=1}^{m} s_{0ik}$). Apresenta-se a seguir, no Algoritmo 16, esta proposta, sendo que a sequência de tarefas é construída considerando w_{ij} mínimos, conforme Mirabi (2011).

Algoritmo 16: Método PESO1.

Entrada: Número de tarefas (n), número de máquinas (m), tempo de processamento (p_{ik} da tarefa i na máquina k) e tempo de *setup* inicial (s_{0ik} da tarefa i na máquina k);

início

  para i=*1* **até** n **faça**

    $z_i = \sum_{k=1}^{m} s_{0ik} + \frac{\sum_{k=1}^{m} p_{ik}}{m}$ onde s_{0ik} é o tempo de *setup* para processar a tarefa i como primeira tarefa na máquina k;

  fim

  para $i = 1$ **até** $n - 1$ **faça**

    para $j = 2$ **até** n **faça**

      se $i \neq j$ **então**

        $w_{ij} = z_j - z_i$

      fim

    fim

  fim

  Escolha aleatoriamente uma tarefa e coloque-a como primeira tarefa da sequência;

  para $a = 2$ **até** n **faça**

    Selecione uma tarefa j, ainda não sequenciada, que tenha w_{ij} mínimo e sequencie-a (onde i é a última tarefa que foi sequenciada);

  fim

  Saída: Sequência de tarefas;

  Busca de soluções vizinhas;

  Saída: Melhor sequência de tarefas.

fim

No final do algoritmo, após a busca por soluções vizinhas, a saída será a melhor sequência de tarefas, dentre todas as sequências. A seguir, apresenta-se o método PESO2, que é uma modificação do apresentado acima, com uma consideração dos tempos de *setup* no cálculo do peso w_{ij}.

4.2.6 PESO2

O método PESO2 é uma modificação do método PESO1. Propõe-se acrescentar uma consideração a respeito do tempo de *setup* no cálculo dos pesos w_{ij}. Nos métodos PALMER2 e NEH2, propostos anteriormente nas Subseções 4.2.2 e 4.2.4, fez-se considerações a respeito dos tempos de *setup*. Nestes métodos, utilizou-se o inverso do valor das somas do tempos de *setup*

pois, a construção da sequência de tarefas era feita utilizando a ordem decrescente dos índices calculados (Q_i e T_i). Nos métodos PESO1 e PESO2, para a construção da sequência de tarefas, utiliza-se os menores valores dos pesos w_{ij} calculados, então, não será usado o inverso da soma dos tempos de *setup*.

Apresenta-se a seguir, no Algoritmo 17, esta proposta.

Algoritmo 17: Método PESO2.

Entrada: Número de tarefas (n), número de máquinas (m), tempo de processamento (p_{ik} da tarefa i na máquina k) e tempos de *setup* (s_{ijk} da tarefa i para tarefa j na máquina k);

início

  para *i=1* **até** n **faça**

    $z_i = \sum_{k=1}^{m} s_{0ik} + \frac{\sum_{k=1}^{m} p_{ik}}{m}$ onde s_{0ik} é o tempo de *setup* para processar a tarefa i como primeira tarefa na máquina k;

  fim

  para $i = 1$ **até** $n - 1$ **faça**

    para $j = 2$ **até** n **faça**

      se $i \neq j$ **então**

        $w_{ij} = z_j - z_i + \frac{\sum_{k=1}^{m} s_{ijk}}{m}$

      fim

    fim

  fim

  Escolha aleatoriamente uma tarefa e coloque-a como primeira tarefa da sequência;

  para $a = 2$ **até** n **faça**

    Selecione uma tarefa j, ainda não sequenciada, que tenha w_{ij} mínimo e sequencie-a (onde i é a última tarefa que foi sequenciada);

  fim

  Saída: Sequência de tarefas;

  Busca de soluções vizinhas;

  Saída: Melhor sequência de tarefas.

fim

Para cada sequência de tarefas (i, j), em PALMER1, calculava-se o peso $w_{ij} = z_j - z_i$. No método PALMER2 é feito um acréscimo dos tempos médios de *setup* ($s_{ij\ medio}$) da sequência de tarefas (i, j) em todas as m máquinas, apresentado na Equação (4.5).

$$s_{ij\ medio} = \frac{\sum_{k=1}^{m} s_{ijk}}{m} \tag{4.5}$$

O peso w_{ij}, para cada sequência de tarefas (i, j), será calculado conforme Equação (4.6).

$$w_{ij} = z_j - z_i + \frac{\sum_{k=1}^{m} s_{ijk}}{m} \tag{4.6}$$

Após a busca por soluções vizinhas, no final do algoritmo, a saída será a melhor sequência de tarefas, dentre todas as sequências. A seguir, apresenta-se o funcionamento da busca por soluções vizinhas, utilizada neste trabalho.

4.3 Busca de soluções vizinhas

A busca de soluções vizinhas é um procedimento importante para melhoramento de soluções. A partir de um estudo na vizinhança pode-se melhorar os resultados já obtidos ou simplesmente, gerar mais soluções vizinhas a partir de uma solução já encontrada. Ruiz, Maroto e Alcaraz (2005) resolveram o problema de *flow shop* permutacional, utilizando um método de busca de soluções vizinhas, para gerar uma população inicial. O método proposto por eles consistia em, escolher aleatoriamente duas tarefas e, trocá-las de posição com as duas primeiras tarefas da sequência.

Neste trabalho, será resolvido o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, com a otimização colônia de formigas. A inicialização dos feromônios será feita a partir de uma população inicial de soluções. Para gerar essa população inicial de soluções serão usados os seis métodos propostos na Seção 4.2, anteriormente: PALMER1, PALMER2, NEH1, NEH2, PESO1 e PESO2.

A partir da solução encontrada por cada um desses métodos, será gerado um conjunto de soluções vizinhas a ela. O número total de soluções que irá compor a população inicial será limitado. O processo de geração de soluções vizinhas, consistirá em sortear duas tarefas aleatoriamente e, trocá-las de posição. A seguir, na próxima seção, apresenta-se o processo de inicialização das trilhas de feromônios.

4.4 Inicialização das trilhas de feromônios

Construída a população de soluções iniciais, é necessário inicializar os valores das trilhas de feromônios. Ying e Liao (2004) que, conforme descrito na Subseção 2.5.3.2, afirmam ser os primeiros a aplicar o sistema colônia de formigas (*Ant Colony System* - ACS), em um problema de *flow shop* permutacional, sem considerar os tempos de *setup*, ressaltam que a inicialização das trilhas de feromônios pode ser de forma aleatória.

Gajpal, Rajendran e Ziegler (2006) propuseram o algoritmo PACA, descrito na Subseção 2.4.4, para resolver o problema de *flow shop* permutacional com sequência dependente dos tempos de *setup*, com base na metaheurística colônia de formigas. Inicializaram todas as trilhas

de feromônios com um mesmo valor, o inverso do menor *makespan* encontrado dentre as soluções iniciais geradas.

Mirabi (2011) também resolveu o problema de *flow shop* permutacional, com sequência dependente dos tempos de *setup*, com a metaheurística colônia de formigas, conforme descrito na Subseção 2.4.5. Ele inicializou as trilhas de feromônios com valores diferentes para cada aresta (i, j), levando-se em consideração o valor do peso w_{ij}.

Ahmadizar (2012) considera uma configuração inicial diferente para as trilhas de feromônios. Após gerar uma sequência inicial s, considere C^s_{max} o *makespan* desta sequência inicial. Calcula-se um limite inferior τ_{min} e um limite superior τ_{max} de acordo com as Equações (4.7) e (4.8), onde ρ é chamado taxa de evaporação de feromônios e, U é um parâmetro entre 0 e 1 sendo que, $\rho > U$.

$$\tau_{max} = \frac{1}{\rho\, C^s_{max}} \tag{4.7}$$

$$\tau_{min} = U\, \tau_{max} \tag{4.8}$$

Todas as trilhas de feromônios (i, j), recebem o valor de $\tau_{ij} = \tau_{min}$ e, em seguida, com intuito de guiar a pesquisa para a vizinhança da sequência inicial s, as trilhas que pertencem a sequência inicial, são modificadas conforme Equação (4.9).

$$\tau_{ij} = (1 - \rho)\, \tau_{ij} + \frac{\rho}{C^s_{max}} \tag{4.9}$$

Essa ideia de usar valores limitantes (τ_{min} e τ_{max}) para as trilhas de feromônios, vêm do método MMAS (Max-Min *Ant System*), proposto por Stützle e Hoos (2000), descrito na Subseção 2.5.3.3. A principal contribuição deste método é, evitar que os valores das trilhas de feromônios sejam discrepantes, levando a uma convergência prematura.

Neste trabalho propõe-se inicializar as trilhas de feromônios com valores de τ_{max} e τ_{min}, calculados conforme Equações (4.10) e (4.11), onde n é o número de tarefas e $L^{inicial}_{best}$ é o valor do *makespan* da melhor solução inicial encontrada.

$$\tau_{max} = \frac{n}{L^{inicial}_{best}} \tag{4.10}$$

$$\tau_{min} = \frac{\tau_{max}}{10} \tag{4.11}$$

As trilhas de feromônios correspondentes a melhor solução inicial encontrada, recebem o valor de τ_{max}, já as trilhas de feromônios das outras soluções vizinhas, recebem o valor de $\frac{\tau_{max}}{2}$ e o restante das trilhas são inicializadas com o valor de τ_{min}. Na próxima seção, apresenta-se a regra de transição, utilizada neste trabalho, para a construção das sequências de tarefas.

4.5 Regra de transição

As regras de transição servem para ajudar na decisão, de qual tarefa será anexada na sequência. Ao aplicar repetidamente a regra de transição, constrói-se uma sequência com todas as tarefas. Nesta construção de soluções, pode-se levar em consideração vários aspectos como, a quantidade de feromônios presente na aresta e também, informações do problema que está sendo resolvido.

Ying e Liao (2004) resolveram o problema de *flow shop* permutacional, sem considerar sequência dependente dos tempos de *setup*, com algoritmo baseado em ACO. Utilizaram a regra de transição proposta no método ACS (*Ant Colony System*) (DORIGO; GAMBARDELLA, 1997), que foi exposta na Seção 2.5.3.2.

Gajpal, Rajendran e Ziegler (2006) resolveram o problema de *flow shop* permutacional, considerando sequência dependente dos tempos de *setup* e, Ahmadizar (2012) resolveu o mesmo problema, sem considerar sequência dependente dos tempos de *setup*, ambos com algoritmo baseado em ACO e, utilizando-se da mesma regra de transição. A regra de transição dividia-se em duas, sorteava-se de um número aleatório em (0,1] e, de acordo com o valor sorteado escolhia-se uma das regras. As regras avaliavam somente os valores das trilhas de feromônios (τ_{ij}).

Mirabi (2011) resolveu o problema de *flow shop* permutacional, considerando sequência dependente dos tempos de *setup*, com algoritmo baseado em ACO e, como regra de transição utilizou uma regra semelhante a deste trabalho. A diferença está no que foi considerado como informação heurística (η_{ij}). Neste trabalho faz-se considerações a respeito dos tempos de *setup* nesta informação heurística, como será exposto a seguir.

Na Seção 2.5.3 apresentou-se as três principais métodos abordados, com base na otimização colônia de formigas: AS (*Ant System*) proposto por Dorigo, Maniezzo e Colorni (1991), ACS (*Ant Colony System*) proposto por Dorigo e Gambardella (1997) e MMAS (Max-Min *Ant System*) proposto por Stützle e Hoos (2000). As regras de transição usadas neste trabalho baseiam-se nos algoritmos AS e ACS.

Cada formiga constrói uma rota, ou seja, forma uma sequência de tarefas. A primeira tarefa da sequência de tarefas é escolhida de forma aleatória. Para a formiga f movimentar-se da tarefa i, precisa escolher uma tarefa j, dentre as tarefas ainda não pertencentes a sequência (conjunto Ω). Para escolher a tarefa j utiliza-se uma, dentre duas regras de transição. Para a escolha de qual regra de transição usar, sorteia-se um número aleatório $q \in (0, 1]$ e, compara-se com um parâmetro já definido $q_0 \in (0, 1]$. Se $q \leq q_0$, usa-se a regra de transição apresentada na Equação (4.12) se, $q > q_0$ usa-se Equação (4.13). Neste trabalho considerou-se $q_0 = 0,5$.

$$r_{ij} = \begin{cases} 1 & \text{se } j = \arg\max_{l \in \Omega} \{\tau_{il}\} \qquad \text{se } q \leq q_0 \\ 0 & \text{caso contrário} \end{cases} \tag{4.12}$$

$$r_{ij} = \begin{cases} \dfrac{\tau_{ij}^{\alpha}\,\eta_{ij}^{\beta}}{\sum_{l\in\Omega_f}\tau_{il}^{\alpha}\,\eta_{il}^{\beta}} & \text{se } j \in \Omega_f \qquad \text{se } q > q_0 \\ 0 & \text{caso contrário} \end{cases} \tag{4.13}$$

$$\eta_{ij} = \frac{1}{\sum_{k=1}^{m} s_{ijk}} \tag{4.14}$$

Na Equação (4.12), escolhe-se como próxima tarefa, a que tiver mais feromônios no caminho relativo a tarefa anterior i, ou seja, escolhe-se como próxima tarefa j, a que tiver maior valor τ_{ij}, o que equivale a $r_{ij} = 1$.

A regra de transição apresentada na Equação (4.13) é a regra proposta em AS, considerou-se $\alpha = 1$ e $\beta = 2$, fornecendo maior peso para a informação heurística na escolha da próxima tarefa. A informação heurística considerada é, a soma do inverso dos tempos de *setup* em todas as máquinas, apresentada na Equação (4.14), onde s_{ijk} é o tempo de *setup* da tarefa i, para tarefa j, na máquina k, e m é o número de máquinas.

A ideia de utilizar, o inverso da soma dos tempos de *setup*, como informação heurística, é para fornecer um peso maior, para as sequências de tarefas que têm um tempo de *setup* menor, e consequentemente, fornecer uma contribuição menor para as sequências de tarefas que têm tempos de *setup* grandes. Acredita-se que este trabalho seja o primeiro a propor o uso dos tempos de *setup*, como informação heurística nas regras de transição. A seguir, na próxima seção, apresenta-se a regra de atualização local utilizada neste trabalho.

4.6 Regra de atualização local

A ideia de fazer uma atualização de feromônios durante o percurso de cada formiga (ou no final), o que chamamos neste trabalho de atualização local dos valores dos feromônios, foi proposta por Dorigo e Gambardella (1997), quando propuseram o *Ant Colony System* (ACS), exposto na Seção 2.5.3.2.

A regra de atualização local deste trabalho é a mesma do algoritmo ACS. Todas as formigas, após cada passo da construção da solução, atualizam o valor do feromônio da aresta visitada de acordo com Equação (4.15), onde ρ ($0 \leq \rho \leq 1$) é a taxa de evaporação do feromônio e, τ_0 é o valor inicial do feromônio. Dorigo e Gambardella (1997) sugerem definir τ_0 como um valor pequeno e, caso já tenha-se uma solução aproximada para problema que está sendo resolvido, pode-se usar o inverso deste valor.

$$\tau_{ij} \longleftarrow (1-\rho)\,\tau_{ij} + \rho\,\tau_0 \tag{4.15}$$

Neste trabalho, considerou-se $\rho = 0,1$ e $\tau_0 = \tau_{min}$. A proposta desta atualização é diversificar as soluções visto que, diminui a quantidade de feromônios nas arestas já visitadas, fazendo com que a mesma rota fique menos atrativa para as próximas formigas.

A utilização da regra de atualização local é comum em trabalhos que resolveram o problema de *flow shop* permutacional, considerando ou não a sequência dependente dos tempos de *setup*, com algoritmos baseados em ACO, pode-se citar alguns deles: (YING; LIAO, 2004), (GAJPAL; RAJENDRAN; ZIEGLER, 2006), (MIRABI, 2011) e (AHMADIZAR, 2012). O que muda de um trabalho para outro é o valor atribuído ao parâmetro de evaporação de feromônios (ρ) e, o que é considerado para fazer um acréscimo de feromônios.

Neste trabalho, para fazer um acréscimo de feromônios considerou-se um valor fixo $\tau_0 = \tau_{min}$, que é calculado com base no *makespan* da melhor solução inicial encontrada. A seguir, na próxima seção, apresenta-se a regra de atualização global utilizada neste trabalho.

4.7 Regra de atualização global

Neste trabalho, considera-se como atualização global o fato de atualizar os feromônios, após todas as formigas terem completado uma solução, adicionando feromônios a rota da melhor solução encontrada até o presente momento. A atualização global foi considerada desde o primeiro algoritmo implementado, baseado na otimização colônia de formigas, o *Ant System* (AS), apresentado na Seção 2.5.3.1.

A atualização global utilizada neste trabalho, é baseada nos algoritmos *Ant Colony System* e MAX-MIN *Ant System*, apresentados nas Seções 2.5.3.2 e 2.5.3.3, respectivamente. Na Equação (4.16) apresenta-se essa regra de atualização global, sendo que $\Delta\tau_{ij}^{best}$ é calculado segundo Equação (4.17) onde L_{best} é o valor do *makespan* da melhor solução encontrada até o momento.

$$\tau_{ij} \longleftarrow \begin{cases} (1-\rho)\,\tau_{ij} + \rho\,\Delta\tau_{ij}^{best} & \text{se } ij \text{ pertence a solução da melhor formiga} \\ (1-\rho)\tau_{ij} & \text{caso contrário} \end{cases} \tag{4.16}$$

$$\Delta\tau_{ij}^{best} = \frac{1}{L_{best}} \tag{4.17}$$

O valor do parâmetro ρ deve ser definido de acordo com o problema que será resolvido, pode-se fazer experimentos computacionais para ajudar na escolha. Neste trabalho considerou-se o parâmetro $\rho = 0,1$. Com esta regra de atualização global, evidencia-se que a melhor rota já encontrada recebe um acréscimo de feromônios, de acordo com a qualidade da solução. Quanto menor for o valor do L_{best} (*makespan* da melhor solução já encontrada), maior será a contribuição na trilha de feromônios (τ_{ij}).

Mesmo que a regra de atualização global já tenha sido proposta desde o primeiro algoritmo baseado em ACO, o AS (*Ant System*), apresentado na Seção 2.5.3.1, nem sempre ela é utilizada. Alguns trabalhos como (GAJPAL; RAJENDRAN; ZIEGLER, 2006) e (MIRABI, 2011), que resolveram o problema de *flow shop* permutacional, considerando a sequência dependente dos tempos de *setup*, com algoritmos baseados em ACO, não utilizaram-se desta regra.

Já (YING; LIAO, 2004) e (AHMADIZAR, 2012), que resolveram o problema de *flow shop* permutacional, sem considerar a sequência dependente dos tempos de *setup*, com algoritmos baseados em ACO, utilizaram-se desta atualização. O que muda de um trabalho para outro é o que foi considerado para o acréscimo de feromônios e, o valor atribuído ao parâmetro de evaporação de feromônios (ρ).

Para o acréscimo de feromônios pode-se fazer considerações a respeito da melhor solução já encontrada, como neste trabalho, que considerou o valor inverso do *makespan* da melhor solução encontrada até o momento. Já o parâmetro de evaporação de feromônios (ρ), pode receber valores diferentes em um mesmo trabalho pois, o valor atribuído na regra de atualização global pode ser diferente do atribuído na regra de atualização local. A seguir, na próxima seção, apresenta-se o critério de parada utilizado neste trabalho.

4.8 Critério de parada

Na resolução de um método heurístico ou metaheurístico necessita-se de um critério de parada para o algoritmo. Na Seção 2.5.3 foram expostos os principais algoritmos baseados na metaheurística colônia de formigas e, todos estes algoritmos utilizam como critério de parada um número máximo de iterações. Neste trabalho também considerou-se como critério de parada um número máximo de 100 iterações.

5 Experimentos Computacionais

Para realizar testes com o modelo MFSP e o algoritmo AFSP proposto, gerou-se instâncias de problemas aleatoriamente, considerando os tempos de processamento identicamente distribuídos entre [1,100] e tempos de *setup* identicamente distribuídos entre [1,10] e [1,50]. Os tempos de *setup* foram aumentados, para testar a influência destes nos métodos de solução. O tamanho dos problemas variou com número de tarefas de 5, 10, 20, 50, 100 e 200 e o número de máquinas de 3, 5, 10 e 20. Considerou-se que as máquinas já estariam preparadas, para realizar a primeira tarefa da sequência de tarefas ou seja, os tempos de *setup* para processar a tarefa i como primeira tarefa na máquina k foram considerados nulos equivalentemente, $s_{0ik} = 0$.

Todas os testes computacionais foram executados em um computador com processador Intel Core i5 2.50GHz, com 4GB de memória RAM. A seguir, apresenta-se os resultados obtidos com os experimentos computacionais, na Seção 5.1 apresenta-se os resultados dos experimentos computacionais feitos com o modelo MFSP, na Seção 5.2 apresenta-se os resultados dos experimentos computacionais feitos com o AFSP e na Seção 5.3 são feitas comparações com os resultados obtidos.

5.1 MFSP

Para testar o modelo MFSP, proposto no Capítulo 3, nas Equações (3.1) até (3.12), utilizou-se o otimizador CPLEX *Interactive Optimizer* 12.4.0.0. Durante a busca por soluções, o otimizador CPLEX, aplica dentre outros métodos um método de busca em árvore, esse método consiste em fazer um relaxamento em variáveis inteiras criando um subproblema, as soluções desses subproblemas são apresentadas e a melhor solução é chamada de *Best Bound*. O otimizador retorna o GAP, utilizado para mensurar a qualidade da solução encontrada, esse valor é obtido conforme Equação (5.1), onde *Best Integer* é o melhor valor inteiro encontrado para o problema até a presente iteração.

$$GAP = \frac{Best\ Integer - Best\ Bound}{Best\ Integer} \tag{5.1}$$

Todas as instâncias de problemas geradas aleatoriamente foram testadas com o modelo MFSP proposto, no otimizador CPLEX. Para a otimização fixou-se o tempo máximo de resolução em 3 horas, o que equivale a 10800 segundos. Na Tabela 1, apresenta-se os resultados para problemas com tempos de *setup* em [1,10] e, na Tabela 2 os resultados para problemas com tempos de *setup* em [1,50].

Tarefas	Máquinas	Tempo (seg)	*Makespan*	GAP[a]
5	3	0,16	256	0%
10	3	921,87	568	0,01%
	5	2408,66	736	0,01%
20	3	10800,95	1152	80,12%
	5	10800,07	1455	73,81%
	10	10799,97	1808	59,75%
50	5	10800,54	3152	88,67%
	10	10800,49	3612	82,64%
	20	10799,12	4421	73,69%
100	5	10827,25	6049	93,85%
	10	10818,48	6627	89,69%
	20	10805,75	8063	84,56%
200	10	10786,47	13223	94,77%
	20	10801,00	NES[b]	-

Tabela 1 – Resultado MFSP para problemas com tempos de *setup* em [1,10], com tempo limitado de 3 horas.

[a] Calculado conforme Equação (5.1).
[b] Não encontrou soluções com esta limitação de tempo.

Tarefas	Máquinas	Tempo (seg)	*Makespan*	GAP[a]
5	3	0,19	326	0%
10	3	1067,58	653	0,01%
	5	2023,88	875	0,01%
20	3	10800,03	1346	82,35%
	5	10800,11	1598	76,13%
	10	10800,62	2117	64,71%
50	5	10800,77	3862	90,76%
	10	10808,15	4516	86,12%
	20	10806,94	5635	78,58%
100	5	10808,59	8054	95,38%
	10	10800,72	NES[b]	-
	20	10812,79	NES	-
200	10	10800,15	NES	-
	20	10800,93	NES	-

Tabela 2 – Resultado MFSP para problemas com tempos de *setup* em [1,50], com tempo limitado de 3 horas.

[a] Calculado conforme Equação (5.1).
[b] Não encontrou soluções com esta limitação de tempo.

Observando as Tabelas 1 e 2 evidencia-se pelo valor do GAP que, para problemas pequenos, com 5 ou 10 tarefas, foi possível encontrar soluções de ótima qualidade em menos de 10800 segundos. Observando os valores do GAP, utilizado para mensurar a qualidade da solução encontrada, percebe-se que quão maior é o tamanho do problema, menor é a qualidade da solução encontrada, dentro da limitação de tempo de 10800 segundos.

A limitação de resolver o problema com modelo MFSP proposto, no otimizador CPLEX, está no fato de não conseguir encontrar soluções para problemas maiores dentro do tempo estabelecido. No conjunto de problemas com *setup* em [1,10], o otimizador CPLEX não encontrou nenhuma solução, dentro do tempo estabelecido, para o problema de 200 tarefas com 20 máquinas. Já, no conjunto de problemas com *setup* em [1,50], não foram encontradas soluções para os problemas com 100 e 200 tarefas com 10 e 20 máquinas.

A partir desta limitação, evidencia-se a importância de utilizar métodos como a metaheurística colônia de formigas que, conseguem encontrar boas soluções para problemas grandes, com um tempo razoável. Os resultados obtidos com o algoritmo proposto neste trabalho, no Capítulo 4, baseado na otimização colônia de formigas, serão apresentados a seguir.

5.2 AFSP

O algoritmo foi programado em *Visual Basic*. Utilizou-se como critério de parada um número fixo de 100 iterações e os seguintes valores para os parâmetros: $q_0 = 0,5$; $\alpha = 1$; $\beta = 2$; $\phi = 0,1$; $\rho = 0,1$, $\tau_0 = \tau_{min}$. O número de formigas foi definido como a metade do número de tarefas. Utilizou-se seis diferentes métodos para geração de soluções iniciais, os quais foram descritos anteriormente, no Capítulo 4 e, chamados de: PALMER1, PALMER2, NEH1, NEH2, PESO1 e PESO2.

Cada instância de problema foi testada 1000 vezes. A seguir, apresenta-se a melhor solução inicial encontrada, ou seja, o menor *makespan*, com cada um dos métodos e, o tempo médio para encontrar esta solução inicial. Também, apresenta-se a melhor solução encontrada com a metaheurística colônia de formigas, dentre as 1000 repetições, partindo das soluções iniciais e, o tempo médio (em segundos) para encontrar essa solução. Os resultados do conjuntos de problemas com tempos de *setup* em [1,10] estão nas Tabelas 3, 4, 5, 6, 7 e 8. Já os resultados do conjunto de problemas com tempos de *setup* em [1,50] estão nas Tabelas 9, 10, 11, 12, 13 e 14.

Para todos os conjuntos de teste, fez-se uma análise estatística com os resultados encontrados nas 1000 repetições para avaliar o valor médio e o desvio padrão dentre os valores encontrados para o *makespan*. Estas análises estão apresentadas em anexo nas Tabelas 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31 e 32.

		Sol Inicial PALMER1		Sol Inicial PALMER1 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	264	0,000029	2	256	0,001618
10	3	581	0,000081	5	570	0,013920
	5	773	0,000089	5	749	0,011843
20	3	1200	0,000229	10	1188	0,088282
	5	1514	0,000317	10	1506	0,096349
	10	2011	0,000609	10	1878	0,126879
50	5	3162	0,001805	25	3230	1,641096
	10	3707	0,003514	25	3700	2,080820
	20	4510	0,006133	25	4587	2,221471
100	5	6063	0,007437	50	6150	22,218223
	10	6632	0,016999	50	6689	29,035807
	20	7665	0,022612	50	7878	21,550791
200	10	12477	0,051686	100	12699	222,783913
	20	13649	0,132562	100	14152	354,289759

Tabela 3 – Resultado com solução inicial gerada com PALMER1 com *setup* em [1,10].

		Sol Inicial PALMER2		Sol Inicial PALMER2 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	256	0,000035	2	256	0,001657
10	3	580	0,000095	5	570	0,013761
	5	789	0,000102	5	747	0,011977
20	3	1200	0,000242	10	1186	0,085611
	5	1536	0,000336	10	1508	0,087739
	10	2011	0,000641	10	1872	0,115177
50	5	3150	0,002166	25	3201	1,861573
	10	3696	0,003921	25	3720	2,004457
	20	4510	0,007957	25	4592	2,463468
100	5	6066	0,011173	50	6137	28,904847
	10	6637	0,023192	50	6695	32,492946
	20	7665	0,028488	50	7848	20,813939
200	10	12481	0,057185	100	12666	217,666795
	20	13644	0,109353	100	14158	236,754979

Tabela 4 – Resultado com solução inicial gerada com PALMER2 com *setup* em [1,10].

Nas Tabelas 3 e 4 visualiza-se que, em alguns casos, o valor do *makespan* encontrado pelos métodos geradores de soluções iniciais, PALMER1 e PALMER2, são menores do que, o valor do *makespan* encontrado com AFSP, partindo destas soluções iniciais. Isto ocorre devido ao fato que a solução inicial gerada não é usada explicitamente, a partir dela geram-se soluções vizinhas e, a solução inicial bem como, as soluções vizinhas só são utilizadas para inicializar os feromônios.

No AFSP, a escolha da sequência de tarefas é regida por um sorteio, que define uma regra de transição, conforme apresentado na Seção 4.5. A construção das soluções é guiada pelos valores dos feromônios (τ_{ij}) e pelos valores da informação heurística (η_{ij}). A informação heurística é regida somente pelos tempos de *setup*. É possível afirmar que esses dois métodos, PALMER1 e PALMER2, foram muito eficazes na geração de soluções, neste conjunto dados usado nos testes. Porém, para trabalhos futuros, pode-se testar outros valores de parâmetros para AFSP, outros critérios como informação heurística e, outros conjuntos de dados para teste, que podem trazer melhores soluções.

		Sol Inicial NEH1		Sol Inicial NEH1 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	270	0,000047	2	256	0,001488
10	3	588	0,000067	5	570	0,010332
	5	826	0,000091	5	757	0,010619
20	3	1300	0,000286	10	1178	0,109179
	5	1661	0,000311	10	1497	0,095697
	10	2127	0,000515	10	1881	0,107133
50	5	3359	0,001713	25	3210	1,676710
	10	4028	0,003857	25	3743	2,262063
	20	4648	0,007080	25	4546	2,503283
100	5	6653	0,007093	50	6181	19,269454
	10	6783	0,014623	50	6751	22,926251
	20	8465	0,027027	50	7874	22,987437
200	10	13319	0,053216	100	12726	251,736556
	20	15145	0,137428	100	14196	351,980683

Tabela 5 – Resultado com solução inicial gerada com NEH1 com *setup* em [1,10].

		Sol Inicial NEH2		Sol Inicial NEH2 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	292	0,000046	2	256	0,001785
10	3	667	0,000850	5	568	0,012754
	5	852	0,000117	5	754	0,014157
20	3	1311	0,000262	10	1181	0,094483
	5	1661	0,000374	10	1494	0,101949
	10	2131	0,000714	10	1893	0,127184
50	5	3330	0,002237	25	3186	1,984705
	10	4023	0,003792	25	3697	1,977816
	20	4593	0,007972	25	4584	2,564022
100	5	6653	0,006446	50	6139	18,157355
	10	6783	0,023883	50	6701	22,709611
	20	8472	0,027162	50	7883	20,067847
200	10	13319	0,113979	100	12639	338,416577
	20	15145	0,190118	100	14128	373,472101

Tabela 6 – Resultado com solução inicial gerada com NEH2 com *setup* em [1,10].

		Sol Inicial PESO1		Sol Inicial PESO1 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	282	0,000030	2	256	0,001611
10	3	715	0,000071	5	568	0,011882
	5	888	0,000091	5	748	0,012051
20	3	1333	0,000213	10	1183	0,082308
	5	1662	0,000305	10	1502	0,090503
	10	2165	0,000515	10	1881	0,107014
50	5	3394	0,002072	25	3213	1,990012
	10	4077	0,003636	25	3726	2,005198
	20	4664	0,006963	25	4523	2,493587
100	5	6722	0,007467	50	6150	19,541710
	10	6838	0,012593	50	6680	17,746832
	20	8546	0,025263	50	7884	20,517913
200	10	13428	0,051384	100	12686	220,965947
	20	15236	0,093426	100	14122	229,509639

Tabela 7 – Resultado com solução inicial gerada com PESO1 com *setup* em [1,10].

		Sol Inicial PESO2		Sol Inicial PESO2 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	279	0,000036	2	256	0,001964
10	3	715	0,000075	5	568	0,011485
	5	896	0,000095	5	756	0,011920
20	3	1264	0,000245	10	1183	0,087891
	5	1656	0,000332	10	1502	0,093130
	10	2122	0,001573	10	1848	0,308876
50	5	3355	0,003294	25	3210	2,911030
	10	4020	0,004123	25	3719	2,201643
	20	4683	0,007413	25	4592	2,447747
100	5	6628	0,008337	50	6137	21,796815
	10	6807	0,014096	50	6685	19,886336
	20	8351	0,027141	50	7882	23,970914
200	10	13149	0,062708	100	12720	248,030569
	20	14926	0,119019	100	14226	266,245554

Tabela 8 – Resultado com solução inicial gerada com PESO2 com *setup* em [1,10].

		Sol Inicial PALMER1		Sol Inicial PALMER1 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	363	0,000031	2	326	0,001461
10	3	705	0,000085	5	654	0,012495
	5	902	0,000109	5	883	0,015836
20	3	1518	0,000233	10	1367	0,090351
	5	1856	0,000326	10	1711	0,095471
	10	2421	0,000532	10	2233	0,103553
50	5	4086	0,001995	25	3980	1,932101
	10	4691	0,003630	25	4561	1,979271
	20	5554	0,006314	25	5512	2,080869
100	5	7898	0,006867	50	7339	16,956884
	10	8700	0,014682	50	8249	22,268311
	20	9907	0,025565	50	9655	24,001687
200	10	16188	0,051306	100	15702	222,720000
	20	17759	0,092810	100	17668	223,533673

Tabela 9 – Resultado com solução inicial gerada com PALMER1 com *setup* em [1,50].

		Sol Inicial PALMER2		Sol Inicial PALMER2 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	363	0,000036	2	326	0,001446
10	3	705	0,000086	5	654	0,012608
	5	902	0,000125	5	875	0,014568
20	3	1518	0,000268	10	1367	0,090416
	5	1856	0,000408	10	1707	0,094330
	10	2421	0,000887	10	2272	0,113030
50	5	4133	0,002415	25	3946	1,798863
	10	4691	0,004140	25	4475	1,975194
	20	5554	0,007595	25	5512	2,229621
100	5	7898	0,007800	50	7339	17,062345
	10	8700	0,013865	50	8249	17,123924
	20	9907	0,029723	50	9668	21,547921
200	10	16176	0,058764	100	15698	213,936812
	20	17759	0,102699	100	17656	221,467417

Tabela 10 – Resultado com solução inicial gerada com PALMER2 com *setup* em [1,50].

Anteriormente, comentou-se que nas Tabelas 3 e 4, que apresentam os resultados das soluções iniciais geradas com os métodos PALMER1, PALMER2 e AFSP, nos conjuntos de teste com tempos de *setup* em [1,10], em alguns casos o valor do *makespan* encontrado, somente pelo método de solução inicial, foi menor que o valor encontrado com AFSP, partindo destas soluções iniciais. Isto pode ocorrer especificamente para o conjunto de dados, em que foram realizados os testes, pode ser devido aos valores atribuídos aos parâmetros ou, pelo o que foi considerado como informação heurística, na regra de transição do AFSP.

Porém, para os testes com tempos de *setup* em [1,50], que os resultados são apresentados nas Tabelas 9 e 10, utilizou-se o mesmo conjunto de tempos de processamento e, em nenhum dos conjuntos de teste, os métodos de solução inicial PALMER1 e PALMER2, foram superiores ao AFSP. Isso pode ocorrer devido ao fato de que, é importante fazer considerações de tempos de *setup* nas regras de transição do AFSP, quando os tempos de *setup* são maiores, ou mais significativos, neste caso, como neste caso que podem ser equivalentes a até 50% dos tempos de processamento.

		Sol Inicial NEH1		Sol Inicial NEH1 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	368	0,000032	2	326	0,001694
10	3	758	0,000084	5	654	0,014593
	5	1015	0,000120	5	875	0,015216
20	3	1610	0,000242	10	1373	0,094416
	5	1993	0,000352	10	1721	0,105200
	10	2516	0,000568	10	2262	0,119809
50	5	4321	0,002035	25	3977	1,811590
	10	5135	0,003561	25	4530	2,174516
	20	5813	0,006659	25	5505	2,268233
100	5	8663	0,006528	50	7339	16,264190
	10	8890	0,012396	50	8229	17,308542
	20	10713	0,023467	50	9632	22,888147
200	10	17455	0,049296	100	15663	217,893347
	20	19205	0,090246	100	17670	219,064881

Tabela 11 – Resultado com solução inicial gerada com NEH1 com *setup* em [1,50].

		Sol Inicial NEH2		Sol Inicial NEH2 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	368	0,000035	2	326	0,001586
10	3	758	0,000091	5	659	0,012627
	5	1015	0,000117	5	887	0,013754
20	3	1610	0,000285	10	1367	0,096028
	5	1993	0,000403	10	1716	0,101050
	10	2516	0,000719	10	2234	0,119275
50	5	4321	0,002210	25	3958	1,754734
	10	5135	0,004387	25	4553	2,101510
	20	5813	0,008683	25	5505	2,607362
100	5	8663	0,007942	50	7339	17,263476
	10	8890	0,015306	50	8229	18,734233
	20	10713	0,029988	50	9652	21,677429
200	10	17455	0,069204	100	15755	253,433998
	20	19208	0,127141	100	17650	265,503802

Tabela 12 – Resultado com solução inicial gerada com NEH2 com *setup* em [1,50].

		Sol Inicial PESO1		Sol Inicial PESO1 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	363	0,000033	2	326	0,001399
10	3	784	0,000085	5	654	0,012458
	5	1085	0,000104	5	875	0,013319
20	3	1641	0,000218	10	1367	0,079571
	5	2029	0,000301	10	1696	0,085757
	10	2519	0,000637	10	2240	0,120828
50	5	4386	0,002106	25	3946	1,783184
	10	5241	0,003238	25	4531	1,862762
	20	5880	0,005926	25	5526	2,207933
100	5	8809	0,007458	50	7409	17,525780
	10	8930	0,014722	50	8229	19,874881
	20	10786	0,027493	50	9632	25,647164
200	10	17529	0,049419	100	15722	205,203428
	20	19301	0,093485	100	17619	225,741051

Tabela 13 – Resultado com solução inicial gerada com PESO1 com *setup* em [1,50].

		Sol Inicial PESO2		Sol Inicial PESO2 + AFSP		
Tarefas	Máquinas	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	326	0,000034	2	326	0,001481
10	3	853	0,000088	5	654	0,012554
	5	1111	0,000116	5	875	0,014056
20	3	1669	0,000281	10	1367	0,098006
	5	1968	0,000359	10	1705	0,109682
	10	2496	0,000556	10	2262	0,115897
50	5	4337	0,002138	25	3970	1,763347
	10	4926	0,003855	25	4535	1,934638
	20	5788	0,006358	25	5505	1,968843
100	5	8450	0,008244	50	7339	18,337652
	10	8910	0,015037	50	8229	20,966198
	20	10378	0,027414	50	9677	21,481322
200	10	17263	0,059945	100	15715	249,653412
	20	18915	0,114037	100	17532	250,561143

Tabela 14 – Resultado com solução inicial gerada com PESO2 com *setup* em [1,50].

5.3 Comparação de resultados

Para avaliar o desempenho das estratégias de resolução propostas, utilizou-se um método de análise de perfis de desempenho, proposto por Dolan e More (2002). A avaliação pode ser feita com vários parâmetros, sendo que deve ser avaliado um parâmetro por vez. Neste trabalho os perfis de desempenho serão avaliados quanto a qualidade do valor encontrado para a função objetivo (FO), ou seja, os valores encontrados para o *makespan* (C_{max}) e também, em relação ao tempo de resolução (t).

Seja P o conjunto teste que têm n_p problemas e, S o conjunto dos n_s métodos de resolução. Para cada problema $p \in P$ e um método de resolução $s \in S$ define-se:

FO_{ps} = o valor da função objetivo obtido para o problema p com o método s

t_{ps} = tempo computacional necessário para resolver o problema p com o método s

Inicialmente calcula-se a razão ou, coeficiente de desempenho (r_{ps}) proposto por Dolan e More (2002), apresentado na Equação 5.2.

$$r_{ps} = \begin{cases} \dfrac{t_{ps}}{min\{t_{ps} \forall s \in S\}} & \text{se a avaliação for em relação ao tempo} \\ \dfrac{FO_{ps}}{min\{FO_{ps} \forall s \in S\}} & \text{se a avaliação for em relação ao valor da FO} \end{cases} \tag{5.2}$$

O coeficiente de desempenho mostra o comportamento de um método para a resolução de um determinado problema. Entretanto, deseja-se uma avaliação geral do desempenho deste método, assim define-se na Equação 5.3, o perfil de desempenho $\gamma_s(\omega)$, onde n_p é a quantidade de problemas do conjunto de problemas P e, ω é um parâmetro que varia de [1,r_M).

$$\gamma_s(\omega) = \frac{1}{n_p} \{p \in P \, ; r_{ps} \leq \omega\} \tag{5.3}$$

$\gamma_s(\omega)$ é a probabilidade que a razão de desempenho r_{ps}, do método $s \in S$, esteja dentro de um parâmetro $\omega \in \mathbb{R}$ da melhor razão de desempenho possível, ou seja, a função $\gamma_s(\omega)$ pode ser vista como a distribuição acumulada da razão de desempenho e, reescrita conforme Equação 5.4.

$$\gamma_s(\omega) = \mathrm{P}(r_{ps} \leq \omega \, ; \; 1 \leq s \leq n_s) \tag{5.4}$$

Para comparação entre os métodos, os perfis de desempenho são apresentados em um único gráfico, por curvas que variam em função do parâmetro ω no intervalo [1,r_M). Ou seja, o eixo das abcissas corresponde ao valor de ω e, o eixo das ordenadas corresponde aos valores de $\gamma_s(\omega)$. Para a criação dos gráficos dos perfis de desempenho deste trabalho, utilizou-se a planilha perfis.xls, versão 1.2, descrita por Munari (2009).

Se o valor de r_M for muito grande, pode-se usar uma escala logarítmica para apresentação dos perfis de desempenho no gráfico e, calcular $\gamma_s(\omega)$ pela Equação (5.5), onde o parâmetro ω irá varia no intervalo $[0, r_M)$.

$$\gamma_s(\omega) = \frac{1}{n_p} \{p \in P \, ; \log_2(r_{ps}) \leq \omega\} \tag{5.5}$$

Caso o método de resolução falhar ao resolver o problema p , deve-se usar $r_{ps} = r_M$, onde $r_M \geq r_{ps} \; \forall \; p \in P, \; \forall \; s \in S$. Neste trabalho definirá-se r_M conforme o que propôs Munari (2009), para escala tradicional ($\omega \in [1, r_M)$) têm-se $r_M = 1 + max\{r_{ps} \; \forall p \in P \text{ e } \forall s \in S\}$, para o caso de escala logarítmica, ($\omega \in [0, r_M)$) têm-se $r_M = 1 + max\{log_2(r_{ps}) \; \forall p \in P \text{ e } \forall s \in S\}$.

Na Tabela 15, apresenta-se os valores obtidos para os coeficientes de desempenho (r_{ps}), em relação ao valor da função objetivo, das soluções iniciais geradas antes de aplicar a metaheurística colônia de formigas, para as instâncias de problemas com valores de *setup* gerados em [1,10].

Tar	Maq	PALMER1	PALMER2	NEH1	NEH2	PESO1	PESO2
5	3	**1,0313**	1,0000	1,0547	1,1406	1,1016	1,0898
10	3	1,0017	1,0000	1,0138	**1,1500**	**1,2328**	**1,2328**
	5	1,0000	**1,0207**	1,0686	1,1022	1,1488	1,1591
20	3	1,0000	1,0000	1,0833	1,0925	1,1108	1,0533
	5	1,0000	1,0145	1,0971	1,0971	1,0978	1,0938
	10	1,0000	1,0000	1,0577	1,0597	1,0766	1,0552
50	5	1,0038	1,0000	1,0663	1,0571	1,0775	1,0651
	10	1,0030	1,0000	1,0898	1,0885	1,1031	1,0877
	20	1,0000	1,0000	1,0306	1,0184	1,0341	1,0384
100	5	1,0000	1,0005	1,0973	1,0973	1,1087	1,0932
	10	1,0000	1,0008	1,0228	1,0228	1,0311	1,0264
	20	1,0000	1,0000	1,1044	1,1053	1,1149	1,0895
200	10	1,0000	1,0003	1,0675	1,0675	1,0762	1,0539
	20	1,0004	1,0000	**1,1100**	1,1100	1,1167	1,0940
Maior valor		1,0313	1,0207	1,1100	1,1500	1,2328	1,2328

Tabela 15 – Razão de desempenho em relação ao valor da função objetivo das soluções iniciais com *setup* em [1,10].

Quando tem-se a razão de desempenho $r_{ps} = 1$, significa que na resolução do problema $p \in P$, o menor valor encontrado para a função objetivo foi, com o método $s \in S$. Na Tabela 15, observa-se que os métodos de solução inicial, PALMER1 e PALMER2, encontraram os menores valores para a função objetivo, dentre as instâncias de problemas resolvidos. Os valores em negrito na Tabela 15, representam o maior coeficiente de desempenho (r_{ps}), encontrado para todos os p problemas, resolvidos com o método s. A seguir na Figura 12, apresenta-se o gráfico do perfil de desempenho, dos métodos utilizados para gerar soluções iniciais, para o conjunto de problemas com valores de *setup* gerados em [1,10].

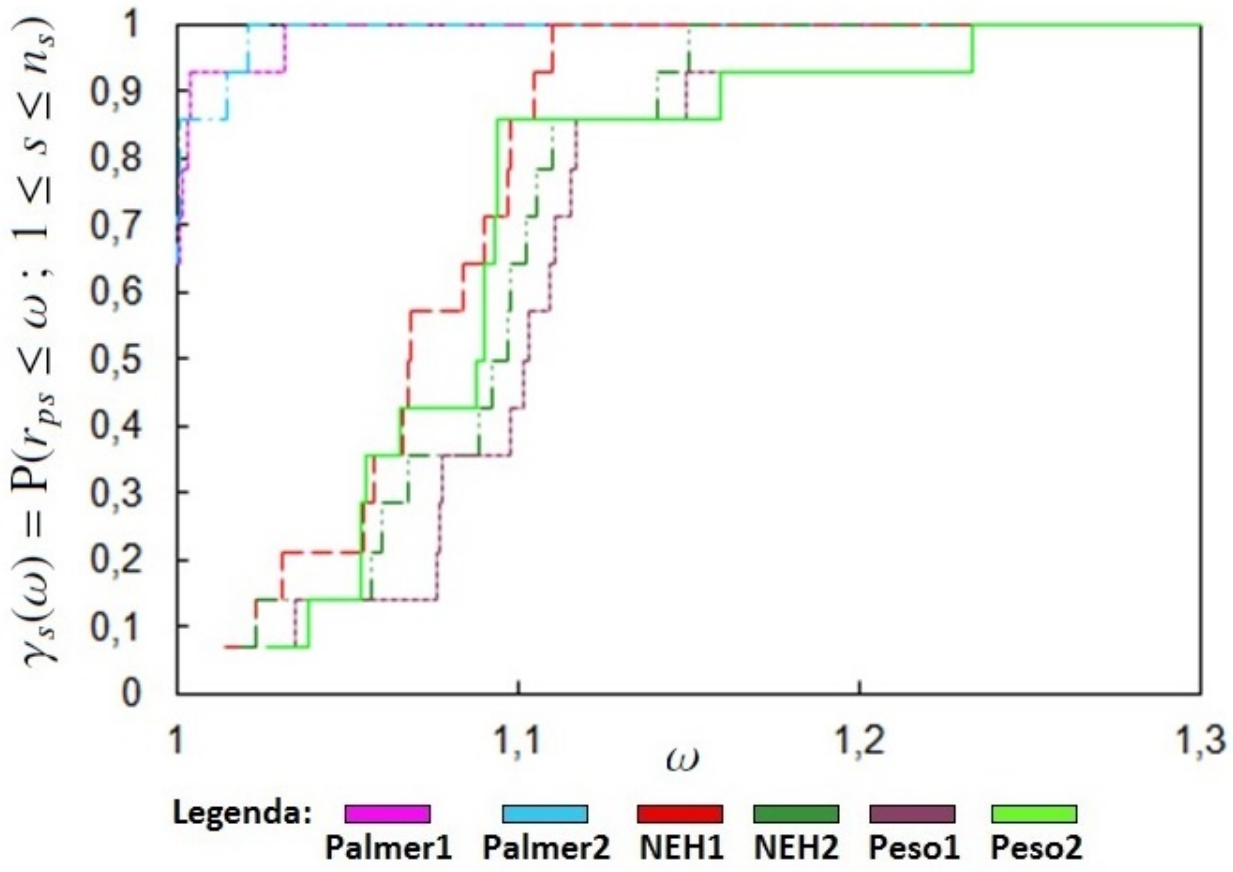

Figura 12 – Gráfico do perfil de desempenho em relação ao valor da função objetivo de todos os métodos de solução inicial com *setup* em [1,10].

Analisando o desempenho final dos métodos no gráfico apresentado na Figura 12 nota-se que o melhor desempenho foi do método PALMER2 que converge um pouco mais rapidamente que o PALMER1. PALMER2 alcança desempenho igual a 1 com $\omega = 1,0207$, enquanto que, PALMER1 alcança com $\omega = 1,0313$. Depois tem-se os métodos NEH1 e NEH2 que atingem o desempenho igual 1 com $\omega = 1,1100$ e $\omega = 1,1500$, respectivamente. Já os métodos PESO1 e PESO2 atingem o desempenho igual 1 juntos em $\omega = 1,2328$.

Evidencia-se que nestes testes os métodos PALMER1 e PALMER2 foram melhores para encontrar soluções iniciais quando comparados aos outros. Comparando NEH1 com NEH2 é possível observar que NEH1 foi superior a NEH2 em mais de 80% dos problemas porém, comparando PESO1 com PESO2, observa-se que o método PESO2 foi superior em mais de 80% dos problemas. Pode-se também afirmar que o método PESO1 teve o pior desempenho em mais de 80% dos problemas, quando comparado com todos os métodos.

O fato do método PALMER2 ter um desempenho final melhor que PALMER1, e PESO2 ter um desempenho melhor que o método PESO1, pode ter sido reflexo da consideração dos tempos de *setup* que não foram feitas para PALMER1 e PESO1. A seguir, na Tabela 16 apresenta-

se os valores das razões de desempenho, em relação ao valor da função objetivo, das soluções iniciais, antes de aplicar a metaheurística colônia de formigas, com os tempos de *setup* em [1,50].

Tar	Maq	PALMER1	PALMER2	NEH1	NEH2	PESO1	PESO2
5	3	**1,1135**	**1,1135**	**1,1288**	**1,1288**	1,1135	1,0000
10	3	1,0000	1,0000	1,0752	1,0752	1,1121	1,2099
	5	1,0000	1,0000	1,1253	1,1253	**1,2029**	**1,2317**
20	3	1,0000	1,0000	1,0606	1,0606	1,0810	1,0995
	5	1,0000	1,0000	1,0738	1,0738	1,0932	1,0603
	10	1,0000	1,0000	1,0392	1,0392	1,0405	1,0310
50	5	1,0000	1,0115	1,0575	1,0575	1,0734	1,0614
	10	1,0000	1,0000	1,0946	1,0946	1,1172	1,0501
	20	1,0000	1,0000	1,0466	1,0466	1,0587	1,0421
100	5	1,0000	1,0000	1,0969	1,0969	1,1153	1,0699
	10	1,0000	1,0000	1,0218	1,0218	1,0264	1,0241
	20	1,0000	1,0000	1,0814	1,0814	1,0887	1,0475
200	10	1,0007	1,0000	1,0791	1,0791	1,0836	1,0672
	20	1,0000	1,0350	1,1192	1,1194	1,1248	1,1023
Maior valor		1,1135	1,1135	1,1288	1,1288	1,2029	1,2317

Tabela 16 – Razão de desempenho em relação ao valor da função objetivo das soluções iniciais com *setup* em [1,50].

Na Tabela 16 observa-se novamente que os métodos de solução inicial, PALMER1 e PALMER2, encontraram os menores valores para a função objetivo, dentre as instâncias de problemas resolvidos. Na Figura 13, exibe-se o gráfico do perfil de desempenho, dos métodos utilizados para gerar soluções iniciais, para todas as instâncias de problemas com valores de *setup* gerados em [1,50].

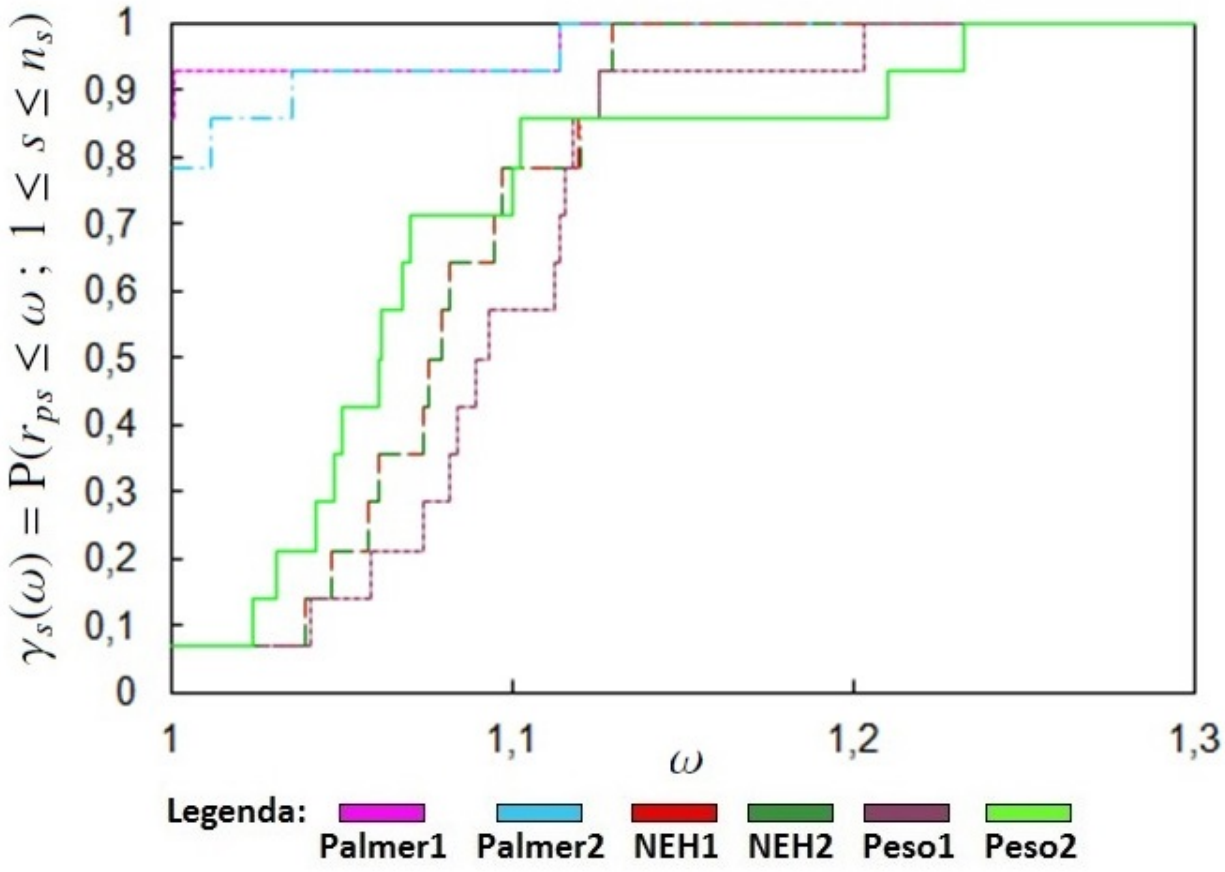

Figura 13 – Gráfico do perfil de desempenho em relação ao valor da função objetivo de todos os métodos de solução inicial com *setup* em [1,50].

No gráfico da Figura 13, observa-se que os métodos PALMER1 e PALMER2 têm comportamento semelhante e superior aos demais métodos, eles atingem desempenho igual a 1 com $\omega = 1,1135$. Observa-se que o método PESO2 atinge o desempenho final em $\omega = 1,2317$ porém, em mais de 70% dos problemas resolvidos foi superior aos métodos NEH1, NEH2 e PESO1. Os métodos NEH1 e NEH2 tiveram o mesmo comportamento e alcançam o desempenho final com $\omega = 1,1288$. O método PESO1 atingiu o desempenho igual a 1 em $\omega = 1,2029$ porém, teve o pior desempenho em mais de 70% dos problemas. O fato do método PESO2 ter apresentado um desempenho melhor que PESO1, pode ter sido pelas considerações a respeito dos tempos de *setup* feitas em PESO2, e que não foram feitas em PESO1.

Na Tabela 17, apresenta-se os valores calculados para a razão de desempenho, em relação ao valor da função objetivo, das soluções encontradas pelos métodos de soluções iniciais propostos com a aplicação da otimização colônia de formigas (ACO) e também, dos valores encontrados para a função objetivo com o modelo MFSP proposto, resolvido pelo otimizador CPLEX.

Para o cálculo de r_M utilizou-se o que propôs Munari (2009), $r_M = 1 + max\{r_{ps}\ \forall p \in P$ e $\forall s \in S\}$. O maior coeficiente de desempenho da Tabela 17 foi $r_{ps} = 1,0407$ então, $r_M = 1 + 1,0470 = 2,0470$.

Tar	Máq	PALMER1 + AFSP	PALMER2 + AFSP	NEH1 + AFSP	NEH2 + AFSP	PESO1 + AFSP	PESO2 + AFSP	MFSP
5	3	1,0313	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
10	3	1,0035	1,0035	1,0035	1,0000	1,0000	1,0000	1,0000
	5	1,0177	1,0149	1,0285	1,0245	1,0163	1,0272	1,0000
20	3	1,0313	1,0295	1,0226	1,0252	1,0269	1,0269	1,0000
	5	1,0351	1,0364	1,0289	1,0268	1,0323	1,0323	1,0000
	10	**1,0387**	1,0354	**1,0404**	**1,0470**	**1,0404**	1,0221	1,0000
50	5	1,0247	1,0155	1,0184	1,0108	1,0194	1,0184	1,0000
	10	1,0244	1,0299	1,0363	1,0235	1,0316	1,0296	1,0000
	20	1,0375	**1,0387**	1,0283	1,0369	1,0231	**1,0387**	1,0000
100	5	1,0167	1,0145	1,0218	1,0149	1,0167	1,0145	1,0000
	10	1,0094	1,0103	1,0187	1,0112	1,0080	1,0088	1,0000
	20	1,0038	1,0000	1,0033	1,0045	1,0046	1,0043	1,0274
200	10	1,0047	1,0021	1,0069	1,0000	1,0037	1,0064	1,0462
	20	1,0021	1,0025	1,0052	1,0004	1,0000	1,0074	$\mathbf{r_M}$
Maior valor		1,0387	1,0387	1,0404	1,0470	1,0404	1,0387	r_M

Tabela 17 – Razão de desempenho das soluções finais, em relação ao valor da função objetivo, com *setup* em [1,10].

Na Tabela 17, observa-se que em alguns problemas os métodos coincidiram de encontrar o menor valor para o *makespan* (função objetivo), exceto o método PALMER1 +AFSP que em nenhum dos problemas encontrou o menor valor. O método NEH1+AFSP atingiu o menor valor em um problema, os métodos PALMER2+AFSP e PESO2+AFSP atingiram em dois problemas, NEH2+AFSP e PESO1+AFSP em três problemas e o modelo MSFP resolvido no CPLEX atingiu em quase todos os problemas, porém, não conseguiu resolver (no tempo limitado de três horas) a maior instância de 200 tarefas e 20 máquinas.

O fato de não conseguir resolver o problema incide em uma penalidade no cálculo do coeficiente de desempenho e, consequentemente, na avaliação geral do perfil de desempenho deste método. Para comparar o desempenho geral, de todos estes métodos em relação ao valor da função objetivo, apresenta-se na Figura 13 o gráfico do perfil de desempenho.

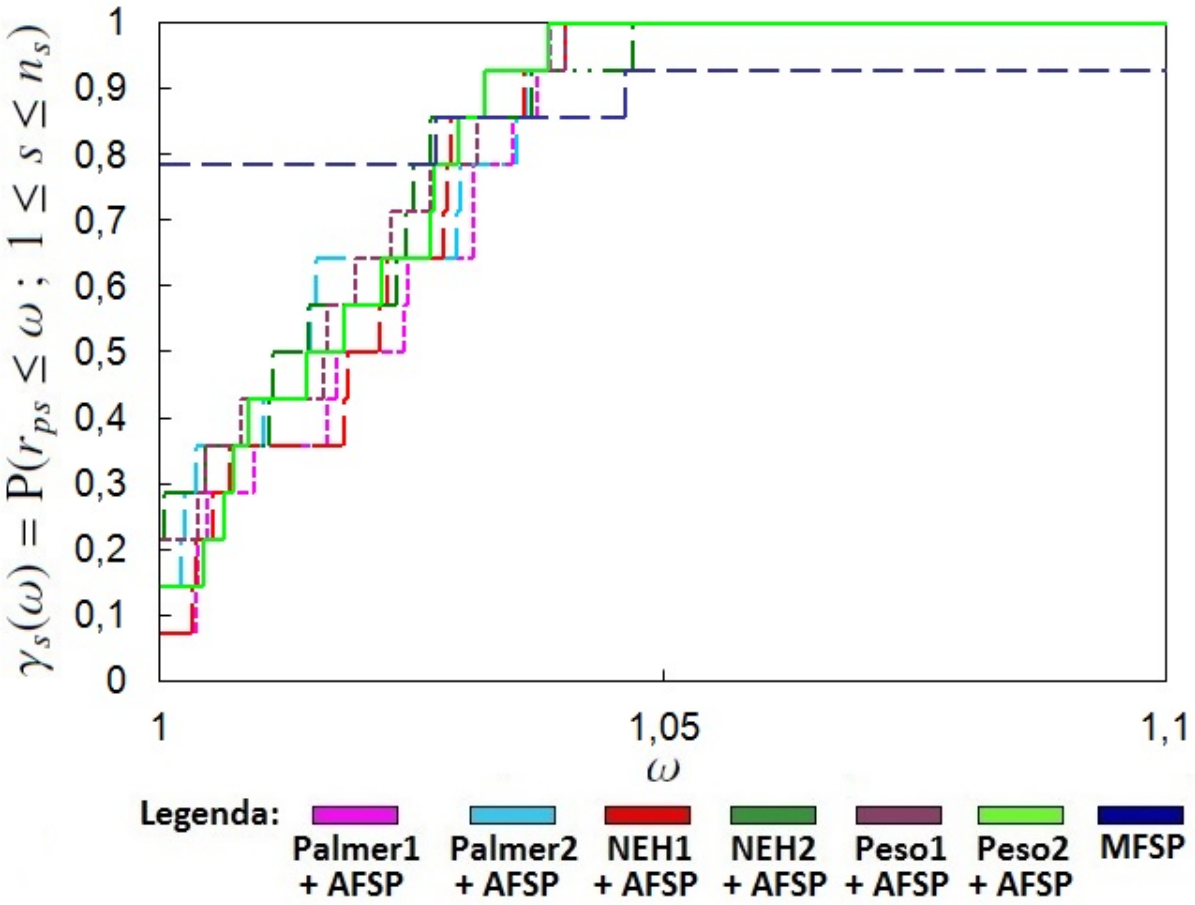

Figura 14 – Gráfico do perfil de desempenho em relação ao valor da função objetivo, de todos os métodos, com *setup* em [1,10].

Observa-se na Figura 14 que, inicialmente o modelo MFSP resolvido pelo CPLEX têm um comportamento superior, enquanto todos os outros métodos, comportam-se similarmente. Porém, todos estes outros métodos atingem o desempenho igual a 1 muito próximos, em $\omega \in [1,0387\ ,\ 1,0470]$, enquanto que, o modelo MFSP resolvido pelo CPLEX só atingirá o desempenho em $r_M = 2,0470$.

Na Tabela 18, exibe-se os resultados dos coeficientes de desempenho (r_{ps}), em relação ao valor da função objetivo, dos problemas com tempos de *setup* em [1,50], resolvidos com todos os métodos propostos, com a aplicação da otimização colônia de formigas (AFSP) e também, do modelo MFSP proposto, resolvido no otimizador CPLEX. O maior coeficiente de desempenho encontrado foi $r_{ps} = 1,0770$, assim, definiu-se $r_M = 1 + 1,0770 = 2,0770$.

Tar	Máq	PALMER1 + AFSP	PALMER2 + AFSP	NEH1 + AFSP	NEH2 + AFSP	PESO1 + AFSP	PESO2 + AFSP	MFSP
5	3	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000	1,0000
10	3	1,0015	1,0015	1,0015	1,0092	1,0015	1,0015	1,0000
	5	1,0091	1,0000	1,0000	1,0137	1,0000	1,0000	1,0000
20	3	1,0156	1,0156	1,0201	1,0156	1,0156	1,0156	1,0000
	5	**1,0707**	1,0682	**1,0770**	**1,0738**	**1,0613**	1,0670	1,0000
	10	1,0548	**1,0732**	1,0685	1,0553	1,0581	**1,0685**	1,0000
50	5	1,0306	1,0218	1,0298	1,0249	1,0218	1,0280	1,0000
	10	1,0192	1,0000	1,0123	1,0174	1,0125	1,0134	1,0092
	20	1,0013	1,0013	1,0000	1,0000	1,0038	1,0000	1,0236
100	5	1,0000	1,0000	1,0000	1,0000	1,0095	1,0000	1,0974
	10	1,0024	1,0024	1,0000	1,0000	1,0000	1,0000	$\mathbf{r_M}$
	20	1,0024	1,0037	1,0000	1,0021	1,0000	1,0047	$\mathbf{r_M}$
200	10	1,0025	1,0022	1,0000	1,0059	1,0038	1,0033	$\mathbf{r_M}$
	20	1,0078	1,0071	1,0079	1,0067	1,0050	1,0000	$\mathbf{r_M}$
Maior valor		1,0707	1,0732	1,0770	1,0738	1,0613	1,0685	r_M

Tabela 18 – Razão de desempenho das soluções finais, em relação ao valor da função objetivo, com *setup* em [1,50].

Na Tabela 18, evidencia-se que para este conjunto de problemas com tempos de *setup* em [1,50], quando comparado com o conjunto com tempos de *setup* em [1,10], o modelo MFSP resolvido com o otimizador CPLEX foi menos eficiente, encontrou a melhor solução somente para as instâncias de problemas menores e, não conseguiu encontrar solução, no tempo limitado de três horas, para as maiores.

O método NEH1+AFSP e o modelo MFSP resolvido com o otimizador Cplex, foram os que mais vezes encontram a melhor solução, totalizando sete, seguidos pelo método PESO2+AFSP, que encontrou a melhor solução para 6 instâncias de problemas. Já PALMER2+AFSP, NEH2+AFSP e PESO1+AFSP encontraram quatro vezes e PALMER1+AFSP somente duas. Para comparar o desempenho geral, em relação ao *makespan*, de todos esses métodos, apresenta-se na Figura 15 o gráfico do perfil de desempenho.

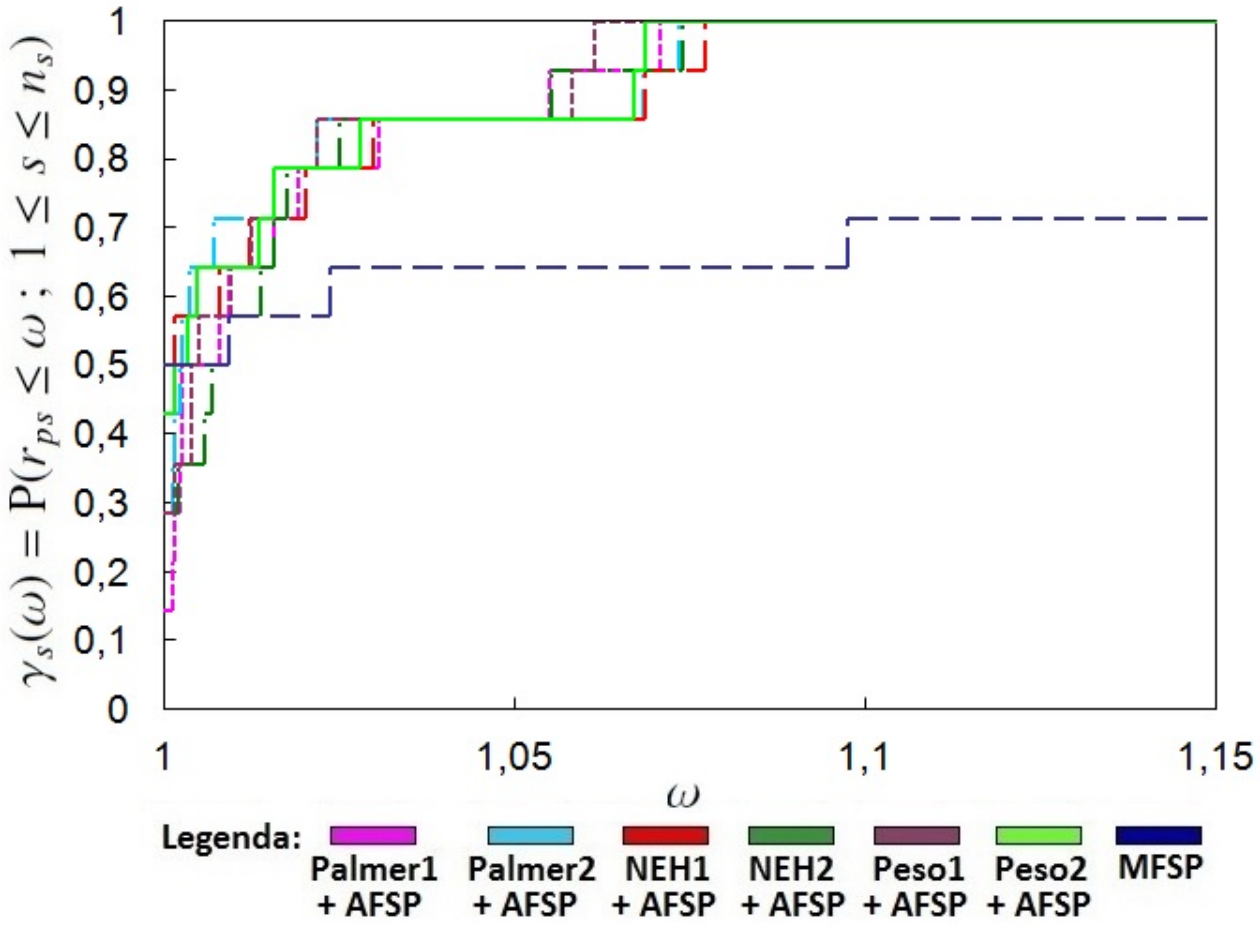

Figura 15 – Gráfico do perfil de desempenho em relação ao valor da função objetivo, de todos os métodos, com *setup* em [1,50].

Na Figura 15, observa-se que inicialmente, o método PALMER1+AFSP têm desempenho inferior, quando comparado aos outros métodos, que comportam-se similarmente. Porém, todos os métodos com a otimização colônia de formigas, atingem o desempenho igual a 1 muito próximos em $\omega \in [1,0613\ ,\ 1,0770]$ enquanto que, o modelo MFSP resolvido pelo otimizador CPLEX, só atingirá o desempenho em $r_M = 2,0770$.

Pode-se observar que MFSP e NEH1+AFSP encontraram a melhor solução para 50% dos problemas, PESO2+AFSP encontrou a melhor solução para 43% dos problemas, PALMER2+AFSP, NEH2+AFSP e PESO1+AFSP encontraram a melhor solução para quase 30% dos problemas. PALMER1+AFSP encontrou a melhor solução para aproximadamente 15% dos problemas. O fato do método PALMER1+AFSP não encontrar boas soluções pode ser por conta de um convergência prematura pois, as soluções iniciais geradas com PALMER1 eram superiores a quase todas as outras soluções geradas com os outros métodos. Este comportamento deve ser avaliado em trabalhos futuros porém, vale ressaltar que, o método PESO1 que encontrou as piores soluções iniciais quando combinado com o AFSP trouxe boas soluções.

Na Tabela 19, exibe-se os resultados dos coeficientes de desempenho (r_{ps}) em relação ao tempo de resolução, dos problemas com tempos de *setup* em [1,10], para os métodos soluções iniciais com AFSP e, o MFSP. Considerando uma escala normal os valores de r_{ps} foram considerados grandes, optou-se pela representação em escala logarítmica. Neste caso, na Tabela 19 apresenta-se os valores de $log_2(r_{ps})$.

Quando opta-se por representar em escala logarítmica, quando têm-se $r_{ps} = 0$, significa que na resolução do problema $p \in P$, o menor tempo foi obtido com o método $s \in S$. O maior coeficiente de desempenho encontrado foi $r_{ps} = 17,7912$ assim, definiu-se $r_M = 17,7912 + 1 = 18,7912$.

Tar	Máq	PALMER1 + AFSP	PALMER2 + AFSP	NEH1 + AFSP	NEH2 + AFSP	PESO1 + AFSP	PESO2 + AFSP	MFSP
5	3	3,4428	0,1552	0,0000	0,2625	0,1146	0,4004	6,7486
10	3	0,4300	0,4135	0,0000	0,3038	0,2017	0,1526	16,4452
	5	0,1574	0,1736	0,0000	0,4149	0,1825	0,1667	17,7912
20	3	0,1011	0,0568	0,4076	0,1990	0,0000	0,0947	17,0017
	5	0,1351	0,0000	0,3154	0,2166	0,0447	0,0860	16,9094
	10	0,4069	0,2673	0,0000	0,4104	0,1613	0,7314	16,7841
50	5	**3,9372**	**4,1190**	0,0000	**4,2114**	**4,2153**	**4,7641**	16,6213
	10	0,3115	0,2576	0,0000	0,2383	0,2581	0,3929	12,6532
	20	0,0000	0,1492	0,0261	0,2069	0,1667	0,1399	12,2471
100	5	3,1499	3,5294	0,0000	2,8587	2,9647	3,0820	12,0786
	10	0,7103	0,8726	0,1188	0,3557	0,0000	0,1642	9,2517
	20	0,1029	0,0527	0,1960	0,0000	0,0320	0,2564	9,0727
200	10	0,0335	0,0000	0,2098	0,6367	0,0217	0,1884	5,6310
	20	0,6264	0,0448	**0,6169**	0,7024	0,0000	0,2142	$\mathbf{r_M}$
Maior valor		3,9372	4,1190	0,6169	4,2114	4,2153	4,7641	r_M

Tabela 19 – Razão de desempenho das soluções finais, em relação ao tempo, com *setup* em [1,10].

Na Tabela 19, observa-se que o método NEH1+AFSP, utilizou o menor tempo de execução, em sete instâncias de problemas, seguido do método PESO1+AFSP, que utilizou menor tempo em três instâncias. O método PESO2+AFSP, não utilizou o menor tempo em nenhuma das instâncias. Esperava-se este comportamento pois, a solução inicial deste método, faz considerações a respeito dos tempos de *setup* que, por exemplo, PESO1+AFSP não faz. Para comparar o desempenho geral, em relação ao tempo de todos os métodos, apresenta-se na Figura 16, o gráfico do perfil de desempenho.

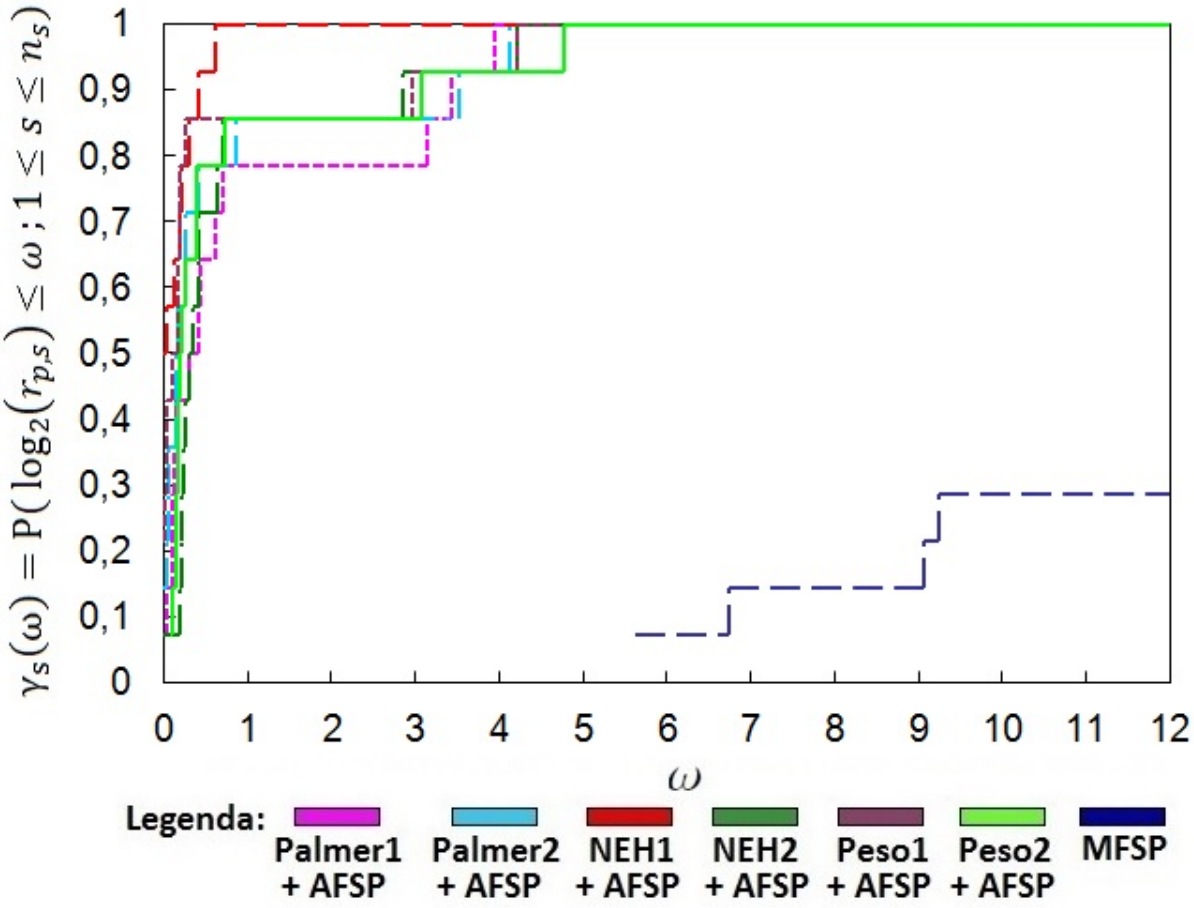

Figura 16 – Gráfico do perfil de desempenho em relação ao tempo, de todos os métodos, com *setup* em [1,10].

Na Figura 16, observa-se que o método NEH1+AFSP é que têm melhor desempenho, atinge desempenho igual a zero em $\omega = 0,6169$, isto porque este método é o que utiliza menor tempo de resolução. Em seguida, têm-se os métodos PALMER1+AFSP, PALMER2+AFSP, NEH2+AFSP, PESO1+AFSP e PESO2+AFSP , que atingem o melhor desempenho em $\omega \in [3,9372\ ,\ 4,7641]$. O modelo MFSP resolvido pelo otimizador CPLEX, como esperava-se, é o que tem menor desempenho em relação ao tempo, só atingirá o desempenho em $r_M = 18,7912$.

Na Tabela 20, exibe-se os resultados da razão de desempenho (r_{ps}), dos métodos de soluções iniciais com o AFSP e o MFSP, em relação ao tempo de resolução dos problemas com tempos de *setup* em [1,50]. O maior coeficiente de desempenho encontrado foi $r_{ps} = 17,2133$ assim, definiu-se $r_M = 17,2133 + 1 = 18,2133$.

Tar	Máq	PALMER1 + AFSP	PALMER2 + AFSP	NEH1 + AFSP	NEH2 + AFSP	PESO1 + AFSP	PESO2 + AFSP	MFSP
5	3	0,0626	0,0477	0,2760	0,1810	0,0000	0,0822	7,0855
10	3	0,0043	0,0173	0,2282	0,0194	0,0000	0,0111	16,3869
	5	0,2497	0,1293	0,1921	0,0464	0,0000	0,0777	17,2133
20	3	0,1833	**0,1843**	0,2468	0,2712	0,0000	0,3006	17,0504
	5	0,1548	0,1375	**0,2948**	0,2367	0,0000	**0,3550**	16,9424
	10	0,0000	0,1263	0,2104	0,2039	0,2226	0,1625	16,6704
50	5	0,1389	0,0358	0,0460	0,0000	0,0232	0,0071	12,5876
	10	0,0875	0,0846	0,2233	0,1740	0,0000	0,0546	12,5024
	20	0,0798	0,1795	0,2042	**0,4052**	0,1653	0,0000	12,4223
100	5	0,0602	0,0691	0,0000	0,0860	0,1078	0,1731	9,3763
	10	**0,3790**	0,0000	0,0155	0,1297	0,2149	0,2921	$\mathbf{r_M}$
	20	0,1601	0,0045	0,0915	0,0131	**0,2557**	0,0000	$\mathbf{r_M}$
200	10	0,1182	0,0601	0,0866	0,3046	0,0000	0,2829	$\mathbf{r_M}$
	20	0,0291	0,0157	0,0000	0,2774	0,0433	0,1938	$\mathbf{r_M}$
Maior valor		0,3790	0,1843	0,2948	0,4052	0,2557	0,3550	r_M

Tabela 20 – Razão de desempenho das soluções finais, em relação ao tempo, com *setup* em [1,50].

Na Tabela 20, observa-se que, para estes problemas com tempo maior de *setup*, o método PESO1+AFSP conseguiu atingir o menor tempo em seis instâncias de problemas. Os métodos NEH1+AFSP e PESO2+AFSP foram mais rápidos em duas instâncias de problemas. Já PALMER1+AFSP, PALMER2+AFSP e NEH2+AFSP atingiram o menor tempo em uma instância. E como já esperado, o modelo MFSP resolvido com o otimizador CPLEX não foi eficiente em relação ao tempo.

Para comparar o desempenho geral de todos os métodos, em relação ao tempo de resolução dos problemas com tempo de *setup* em [1,50], apresenta-se na Figura 17 o gráfico do perfil de desempenho.

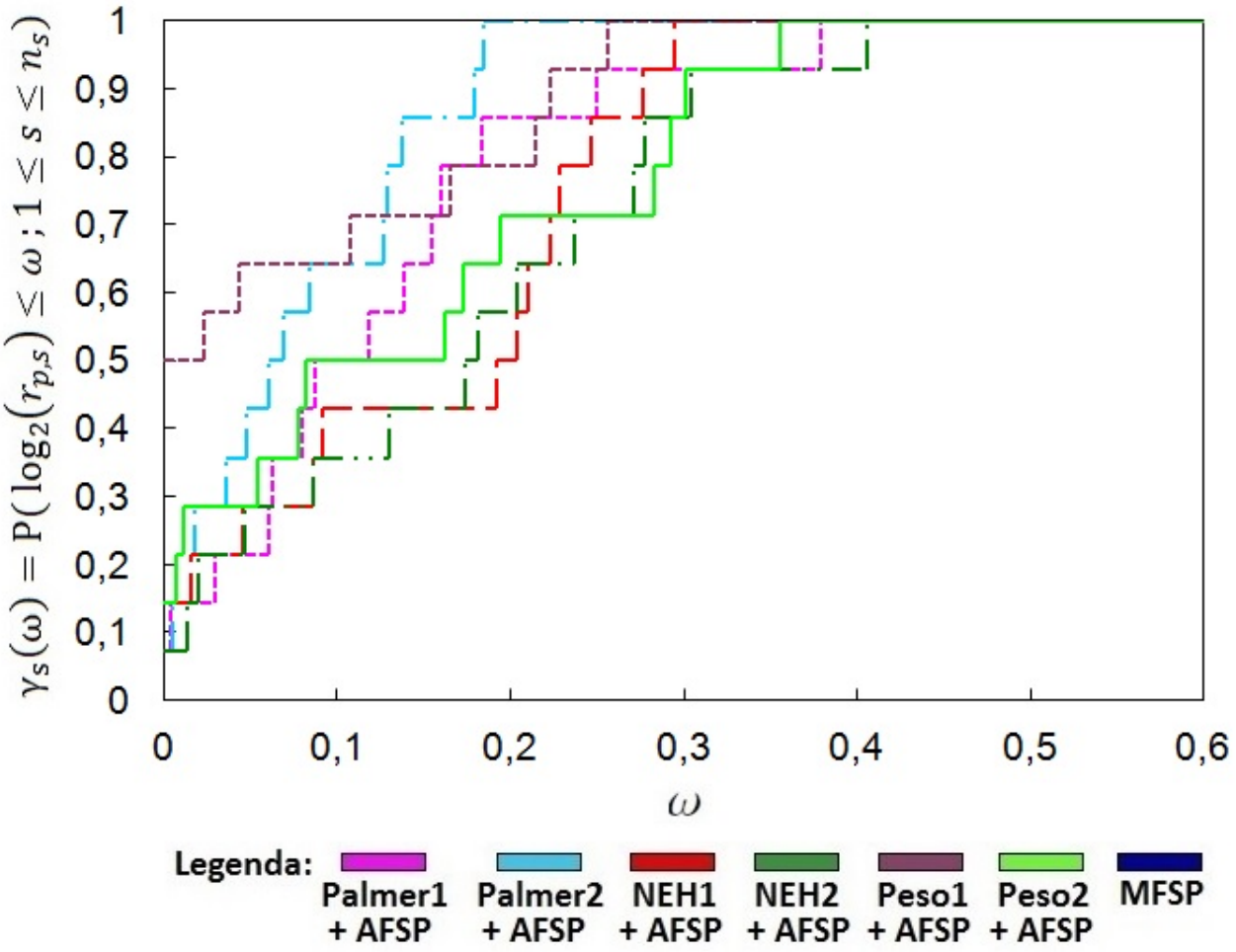

Figura 17 – Gráfico do perfil de desempenho em relação ao tempo, de todos os métodos, com *setup* em [1,50].

No gráfico apresentado na Figura 17, a linha correspondente ao modelo MFSP não aparece. Isto ocorre porque os valores de ω, para o MFSP, são muito grandes. Plotar o gráfico considerando o eixo ω com valores maiores, não permite visualizar o comportamento dos outros métodos, que variam até, aproximadamente $\omega = 0,4$.

Na Figura 17, evidencia-se que o método que obteve melhor desempenho em relação ao conjunto de problemas com *setup* em [1,50], foi o método PALMER2+AFSP, que atinge o desempenho igual a zero com $\omega = 0,1843$. Em seguida tem-se o método PESO1+AFSP, com $\omega = 0,2557$ e, NEH1+AFSP com $\omega = 0,2948$. Os métodos PALMER1+AFSP, NEH2+AFSP e, PESO2+AFSP atingem o desempenho praticamente juntos em $\omega \in [0,3550\ ,\ 0,4052]$. Já o modelo MFSP só atingirá o desempenho igual a zero em $\omega = r_M = 18,2133$.

Com estes resultados, evidencia-se que o modelo (MFSP), resolvido com o otimizador CPLEX, utilizou um tempo maior para resolver problemas de pequeno e médio porte e, teve dificuldade em resolver problemas de grande porte, com a limitação de tempo em três horas. Nas

soluções iniciais, geradas com seis diferentes métodos, o perfil de desempenho mostra que os métodos PALMER1+AFSP e PALMER2+AFSP, foram superiores aos outros com relação ao valor da função objetivo.

Nos casos em que aplicou-se o algoritmo (AFSP), com a inicialização das trilhas de feromônios a partir das soluções iniciais geradas com os seis métodos, verificou-se que nos conjuntos de problemas com tempos de *setup* em [1,10], de acordo com os perfis de desempenho, os métodos PALMER1+AFSP, PALMER2+AFSP e PESO2+AFSP, foram superiores em relação ao valor da função objetivo e, o método NEH1+AFSP foi o mais rápido. Já, nos conjuntos de problemas com tempos de *setup* em [1,50], de acordo com os perfis de desempenho, os métodos PESO1 e PESO2 foram superiores em relação ao valor da função objetivo e, PALMER2 foi o mais rápido. No Capítulo 6, apresenta-se algumas considerações finais deste trabalho e, sugestões para trabalhos futuros.

6 Considerações finais e sugestões para trabalhos futuros

A proposta inicial deste trabalho, baseava-se no estudo do problema de *flow shop* permutacional. Após revisão de trabalhos já propostos na literatura, percebeu-se um número pequeno de publicações que consideram a sequência dependente dos tempos de *setup* e, um crescente número de publicações para resolver este problema com técnicas enumeração, heurísticas e metaheurísticas. Assim, propôs-se resolver este problema com a metaheurística colônia de formigas.

Uma das propostas consistia em descrever o problema de*flow shop* permutacional, com sequência dependente dos tempos de *setup*, por meio de um modelo matemático, o modelo (MFSP) foi apresentado no Capítulo 3.

Outro objetivo deste trabalho, era testar a influência da consideração dos tempos de *setup* na qualidade de soluções iniciais. Na Seção 4.2 apresentou-se os métodos utilizados para gerar soluções iniciais. Utilizou-se três métodos já propostos na literatura e a partir destes, foi proposto outros três, acrescentando considerações a respeito dos tempos de *setup*.

Para testar a influência dos tempos de *setup*, gerou-se conjuntos de dados aleatórios com tempos de *setup* em [1,10] e [1,50]. O fato de aumentar esses tempos de *setup* foi para testar a influência desta consideração nos métodos. No Capítulo 5, apresentou-se os resultados dos testes computacionais realizados e os gráficos de perfis de desempenho dos métodos. Estes gráficos foram de suma importância para avaliar o comportamento destes métodos.

No conjuntos de problemas com tempos de *setup* em [1,10] o método PALMER2 mostrou-se superior a todos os outros métodos, e o método PESO1 foi o que teve pior desempenho. O método PESO2 mostrou-se superior ao método PESO1. O fato do método PALMER2 ter um desempenho melhor que PALMER1, e o método PESO2 ter um desempenho melhor que PESO1, mostra que as considerações dos tempos de *setup* feitas em PALMER2 e PESO2 foram positivas.

Comparando os resultados obtidos com os seis métodos, no conjunto de problemas com tempos de *setup* em [1,50], os métodos PALMER1 e PALMER2 mostraram-se superiores aos outros métodos enquanto que, o método PESO1 novamente foi o que teve o pior desempenho em mais de 70% dos problemas. O método PESO2 mostrou-se superior aos métodos NEH1, NEH2 e PESO1 em mais de 70% dos problemas. Acredita-se que as considerações dos tempos de *setup* feitas em PESO2 contribuíram para que este método encontrasse boas soluções.

Outra proposta deste trabalho era desenvolver um algoritmo baseado na metaheurística colônia de formigas, com considerações dos tempos de *setup* nas regras de transição. Este algoritmo (AFSP) foi apresentado no Capítulo 4. Para inicializar os feromônios utilizou-se as

soluções iniciais geradas com e sem considerações a respeitos dos tempos de *setup*. No Capítulo 5 apresentou-se os resultados. Evidenciou-se que os resultados obtidos foram satisfatórios, quando comparados com os resultados obtidos pelo modelo matemático (MFSP) com o otimizados CPLEX.

Para o conjunto de problemas com tempos de *setup* em [1,10], observando o gráfico do perfil de desempenho, todos os métodos com o AFSP e o MFSP apresentaram um comportamento semelhante. Para o conjunto de problemas com tempos de *setup* em [1,50], observando o gráfico do perfil de desempenho, MFSP e NEH1+AFSP encontraram a melhor solução para 50% dos problemas, PESO2+AFSP encontrou a melhor solução para 43% dos problemas, PALMER2+AFSP, NEH2+AFSP e PESO1+AFSP encontraram a melhor solução para quase 30% dos problemas. PALMER1+AFSP encontrou a melhor solução para aproximadamente 15% dos problemas. Este comportamento sugere que inicializar as trilhas de feromônios com soluções iniciais muito boas, como é o caso de PALMER1, pode levar a uma convergência prematura. Este estudo é uma sugestão para trabalhos futuros.

Não foi possível fazer-se comparações com outros algoritmos, baseados na metaheurística colônia de formigas, que não considerassem o tempo de *setup* nas regras de transição. Esta comparação também é deixada como sugestão para trabalhos futuros.

Referências

AHMADIZAR, F. A new ant colony algorithm for makespan minimization in permutation flow shops. **Computers & Industrial Engineering**, v. 63, n. 2, p. 355–361, 2012.

ALLAHVERDI, A.; NG, C. T.; CHENG, T. C. E.; KOVALYOV, M. Y. A survey of scheduling problems with setup times or costs. **European Journal of Operational Research**, v. 187, p. 985–1032, 2008.

ARENALES, M.; ARMENTANO, V.; MORABITO, R.; YANASSE, H. H. **Pesquisa operacional para cursos de engenharia**. [S.l.]: Elsevier, 2007.

BAKER, K. R.; TRIETSCH, D. **Principles of sequencing and scheduling**. New Jersey: John Wiley & Sons, 2009.

BRANCO, F. J. C. **Um novo método heurístico construtivo de alto desempenho para o problema no-idle flow shop**. Tese de Doutorado em Engenharia de Produção — Universidade de São Paulo, São Carlos, SP, 2011.

BRUCKER, P. **Scheduling Algorithms**. 5. ed. Berlin: Springer, 2007.

DANNEBERG, D.; TAUTENHAHN, T.; WERNER, F. A comparison of heuristic algorithms for flow shop scheduling problems with setup times and limited batch size. **Mathematical and computer modeling**, v. 29, n. 9, p. 101–126, 1999.

DENEUBOURG, J. L.; ARON, S.; GOSS, S.; PASTEELS, J. M. The self-organizing exploratory pattern of the argentine ant. **Journal of insect behavior**, v. 3, n. 2, p. 159–168, 1990.

DOLAN, E. D.; MORE, J. J. Benchmarking optimization software with performance profiles. **Journal Mathematical Programming**, v. 91, n. 2, p. 201–213, 2002.

DORIGO, M.; CARO, G. D. The ant colony optimization meta-heuristic. In: CORNE, D. AND DORIGO, M. AND GLOVER, F. **New Ideas in Optimization**. New York: McGraw-Hill, 1999. p. 11–32.

DORIGO, M.; CARO, G. D.; GAMBARDELLA, L. Ant algorithms for discrete optimization. **Artificial Life**, v. 5, p. 137–172, 1999.

DORIGO, M.; GAMBARDELLA, L. M. Ant colony system: a cooperative learning approach to the traveling salesman problem. **IEEE Transactions on Evolutionary Computation**, v. 1, p. 53–66, 1997.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. **Positive feedback as a search strategy**. Technical report 91-016. Milan, Italy: Politecnico di Milano, Dipartimento di Elettronica, 1991.

DORIGO, M.; MANIEZZO, V.; COLORNI, A. Ant system: optimization by a colony of cooperating agents. **IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics**, v. 26, p. 29–41, 1996.

DORIGO, M.; SOCHA, K. **An introduction to ant colony optimization**. Bruxelles, Belgium, 2006.

GAJPAL, Y.; RAJENDRAN, C.; ZIEGLER, H. An ant colony algorithm for scheduling in flowshops with sequence-dependent setup time of jobs. **The International Journal of Advanced Manufacturing Technology**, v. 30, p. 416–424, 2006.

GAREY, M. R.; JOHNSON, D. S. **Computers and Intractability: A Guide to the Theory of NP-Completeness**. New York, NY, USA: W. H. Freeman & Co., 1979.

GOSS, S.; ARON, S.; DENEUBOURG, J. L.; PASTEELS, J. M. Self-organized shortcuts in the argentine ant. **Naturwissenschaften**, v. 76, p. 579–581, 1989.

IGNALL, E.; SCHRAGE, L. Application of the branch and bound technique to some flow-shop scheduling problems. **Operations Research**, v. 13, n. 3, p. 400–412, 1965.

JOHNSON, S. M. Optimal two and three-stage production schedules with setup times included. **Naval Research Logistics Quarterly**, p. 61–68, 1954.

LENSTRA, J. K.; RINNOOY, A. H. G. K. Complexity of machine scheduling problems. **Journal Mathematical Programming**, v. 1, p. 343–362, 1977.

LI, X.; ZHANG, Y. Adaptive hybrid algorithms for the sequence-dependent setup time permutation flow shop scheduling problem. **IEEE Transactions on Automation Science and Engineering**, v. 9, n. 3, p. 578–595, 2012.

LUO, X.; CHU, C. A branch-and-bound algorithm of the single machine schedule with sequence-dependent setup times for minimizing maximum tardiness. **European journal of operational research**, v. 180, p. 68–81, 2007.

LUSTOSA, L. J.; MESQUITA, M. A.; QUELHAS, O. L. G.; OLIVEIRA, R. J. **Planejamento e controle da produção**. Rio de Janeiro: Elsevier, 2008.

MIRABI, M. Ant colony optimization technique for the sequence-dependent flowshop scheduling problem. **The International Journal of Advanced Manufacturing Technology**, v. 55, p. 317–326, 2011.

MIRABI, M. A novel hybrid genetic algorithm to solve the sequence-dependent permutation flow-shop scheduling problem. **The international journal of advanced manufacturing technology**, v. 71, p. 429–437, 2014.

MUNARI, P. A. **Comparação de softwares científicos utilizando perfis de desempenho: automatização dos cálculos pela planilha perfis.xls**. Relatório técnico 344. São Carlos, SP: Instituto de Ciências Matemáticas e de Computação - Universidade de São Paulo, 2009. Disponível em: http://www.icmc.usp.br/Portal/conteudo/1008/737/relatorios-tecnicos.

NAWAZ, M.; ENSCORE, E. E. J.; HAM, I. A heuristic algorithm for the m-machine, n-job flowshop sequencing problem. **Omega**, v. 11, p. 91–95, 1983.

NEUFELD, J. S.; GUPTA, J. N. D.; BUSCHER, U. A comprehensive review of flowshop group scheduling literature. **Computers & operations research**, v. 70, p. 56–74, 2016.

PALMER, D. S. Sequencing jobs through a multistage process in the minimum total time: a quick method of obtaining a near-optimum. **Operational Research Quarterly**, v. 16, p. 101–107, 1965.

RAJENDRAN, C.; ZIEGLER, H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. **European Journal of Operational Research**, v. 155, p. 426–438, 2004.

RíOS-MERCADO, R. Z.; BARD, J. F. A branch-and-bound algorithm for permutation flow shops with sequence-dependent setup times. **IIE Transactions**, v. 31, n. 8, p. 721–731, 1999.

RUIZ, R.; MAROTO, C.; ALCARAZ, J. Solving the flowshop scheduling problem with sequence dependent setup times using advanced metaheuristics. **European Journal of Operational Research**, v. 165, p. 34–54, 2005.

SARAVANAN, M.; VIJAYAKUMAR, S. J. D.; SRINIVASAN, R. A multicriteria permutation flow shop scheduling problem with setup times. **International journal of engineering and technology**, v. 6, n. 3, p. 1329–1339, 2014.

SHEN, L.; GUPTA, J. N. D.; BUSCHER, U. Flow shop batching and scheduling with sequence-dependent setup times. **Journal of scheduling**, v. 17, n. 4, p. 353–370, 2014.

SOUZA, E. C. **Programação de tarefas em um flow shop**. Tese de Doutorado em Engenharia — Universidade de São Paulo, São Paulo, 2009.

STüTZLE, T.; HOOS, H. H. Max-min ant system. **Future Generation Computer Systems**, v. 16, n. 8, p. 889–914, 2000.

TUBINO, D. F. **Planejamento e controle da produção: teoria e pratica**. 2. ed. São Paulo: Atlas, 2009.

VANCHIPURA, R.; SRIDHARAN, R. Development and analysis of constructive heuristic algorithms for flow shop scheduling problems with sequence-dependent setup times. **The international journal of advanced manufacturing technology**, v. 67, n. 5, p. 1337–1353, 2013.

YAGMAHAN, B.; YENISEY, M. M. Ant colony optimization for multi-objective flow shop scheduling problem. **Computers & Industrial Engineering**, v. 54, p. 411–420, 2008.

YAGMAHAN, B.; YENISEY, M. M. A multi-objective ant colony system algorithm for flow shop scheduling problem. **Expert Systems with Applications**, v. 37, p. 1361–1368, 2010.

YING, K. C.; LIAO, C. J. An ant colony system for permutation flow-shop sequencing. **Computers & Operations Research**, v. 31, n. 5, p. 791–801, 2004.

A Anexo 1

		Sol Inicial PALMER1		Sol Inicial PALMER1 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	269,17 ± 6,62	0,000029	2	263,34 ± 6,94	0,001618
10	3	712,34 ± 10,01	0,000081	5	705,47 ± 8,50	0,013920
	5	776,66 ± 5,98	0,000089	5	798,33 ± 14,47	0,011843
20	3	1220,00 ± 12,28	0,000229	10	1219,60 ± 9,44	0,088282
	5	1556,90 ± 12,48	0,000317	10	1544,53 ± 14,79	0,096349
	10	2026,06 ± 12,95	0,000609	10	1951,44 ± 17,53	0,126879
50	5	3182,11±15,52	0,001805	25	3321,79±22,79	1,641096
	10	3755,36±21,28	0,003514	25	3864,74±34,16	2,080820
	20	4527,79±9,30	0,006133	25	4719,90±30,63	2,221471
100	5	6114,64±15,29	0,007437	50	6260,13±24,80	22,218223
	10	6672,81±20,29	0,016999	50	6814,01±30,47	29,035807
	20	7688,80±10,47	0,022612	50	8024,89±29,33	21,550791
200	10	12526,33±18,70	0,051686	100	12835,91±47,64	222,783913
	20	13698,70±19,50	0,132562	100	14307,52±50,81	354,289759

Tabela 21 – Resultado com solução inicial gerada com PALMER1 com *setup* em [1,10].

		Sol Inicial PALMER2		Sol Inicial PALMER2 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	256,88±1,79	0,000035	2	263,58±6,73	0,001657
10	3	580,99±2,17	0,000095	5	586,91±7,69	0,013761
	5	791,87±4,09	0,000102	5	798,33±15,10	0,011977
20	3	1220,10±12,57	0,000242	10	1219,30±9,45	0,085611
	5	1556,35±12,49	0,000336	10	1543,72±14,38	0,087739
	10	2026,41±12,99	0,000641	10	1951,31±17,90	0,115177
50	5	3178,49±20,58	0,002166	25	3320,97±23,43	1,861573
	10	3743,12±26,53	0,003921	25	3863,46±34,02	2,004457
	20	4526,85±9,35	0,007957	25	4717,30±32,57	2,463468
100	5	6139,38±15,63	0,011173	50	6261,54±25,21	28,904847
	10	6687,78±19,58	0,023192	50	6813,77±31,32	32,492946
	20	7688,79±10,44	0,028488	50	8023,77±31,28	20,813939
200	10	12522,96±20,04	0,057185	100	12830,98±46,78	217,666795
	20	13697,92±21,37	0,109353	100	14319,46±43,55	236,754979

Tabela 22 – Resultado com solução inicial gerada com PALMER2 com *setup* em [1,10].

		Sol Inicial NEH1		Sol Inicial NEH1 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	286,38±12,17	0,000047	2	265,67±6,24	0,001488
10	3	602,28±17,02	0,000067	5	587,85±10,75	0,010332
	5	856,23±17,95	0,000091	5	799,44±17,11	0,010619
20	3	1306,21±6,33	0,000286	10	1218,89±9,86	0,109179
	5	1681,28±18,64	0,000311	10	1542,89±15,05	0,095697
	10	2154,22±19,30	0,000515	10	1953,82±15,47	0,107133
50	5	3383,08±15,13	0,001713	25	3318,78±23,52	1,676710
	10	4047,04±11,40	0,003857	25	3867,53±28,89	2,262063
	20	4679,69±12,02	0,007080	25	4711,06±33,27	2,503283
100	5	6680,67±15,20	0,007093	50	6260,34±24,27	19,269454
	10	6826,05±17,22	0,014623	50	6814,89±29,55	22,926251
	20	8517,78±23,12	0,027027	50	8024,19±30,84	22,987437
200	10	13363,93±14,31	0,053216	100	12839,68±44,70	251,736556
	20	15182,45±18,03	0,137428	100	14316,12±48,12	351,980683

Tabela 23 – Resultado com solução inicial gerada com NEH1 com *setup* em [1,10].

		Sol Inicial NEH2		Sol Inicial NEH2 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	295,57±6,02	0,000046	2	265,98±5,89	0,001785
10	3	707,75±34,82	0,000850	5	588,04±12,18	0,012754
	5	874,44±22,64	0,000117	5	800,63±16,67	0,014157
20	3	1316,94±6,83	0,000262	10	1219,01±9,68	0,094483
	5	1679,26±18,26	0,000374	10	1542,87±13,89	0,101949
	10	2159,97±19,47	0,000714	10	1951,67±16,07	0,127184
50	5	3355,69±14,31	0,002237	25	3319,33±23,97	1,984705
	10	4042,99±10,96	0,003792	25	3867,95±30,05	1,977816
	20	4631,86±19,08	0,007972	25	4709,97±33,40	2,564022
100	5	6714,60±15,82	0,006446	50	6261,47±25,02	18,157355
	10	6817,24±17,22	0,023883	50	6814,40±30,39	22,709611
	20	8513,60±17,68	0,027162	50	8025,68±29,44	20,067847
200	10	13366,92±13,64	0,113979	100	12836,21±47,34	338,416577
	20	15180,34±16,46	0,190118	100	14312,08±44,92	373,472101

Tabela 24 – Resultado com solução inicial gerada com NEH2 com *setup* em [1,10].

		Sol Inicial PESO1		Sol Inicial PESO1 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	323,50±23,07	0,000030	2	264,16±6,48	0,001611
10	3	784,32±39,63	0,000071	5	590,40±13,87	0,011882
	5	942,42±30,90	0,000091	5	801,34±17,74	0,012051
20	3	1387,40±24,32	0,000213	10	1217,85±10,06	0,082308
	5	1746,74±35,78	0,000305	10	1543,05±14,21	0,090503
	10	2245,94±38,15	0,000515	10	1951,45±16,30	0,107014
50	5	3484,52±41,31	0,002072	25	3319,37±23,76	1,990012
	10	4147,74±37,23	0,003636	25	3861,50±32,38	2,005198
	20	4764,58±39,13	0,006963	25	4708,42±33,76	2,493587
100	5	6813,40±32,84	0,007467	50	6260,64±24,50	19,541710
	10	6927,40±35,78	0,012593	50	6814,90±29,18	17,746832
	20	8650,09±42,38	0,025263	50	8024,19±30,69	20,517913
200	10	13499,57±37,84	0,051384	100	12834,13±48,04	220,965947
	20	15350,92±47,75	0,093426	100	14311,26±48,74	229,509639

Tabela 25 – Resultado com solução inicial gerada com PESO1 com *setup* em [1,10].

		Sol Inicial PESO2		Sol Inicial PESO2 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	321,98±22,34	0,000036	2	264,30±6,23	0,001964
10	3	784,54±38,86	0,000075	5	590,37±13,60	0,011485
	5	951,41±41,63	0,000095	5	801,88±17,63	0,011920
20	3	1389,16±51,10	0,000245	10	1218,23±10,36	0,087891
	5	1757,84±42,83	0,000332	10	1542,06±14,35	0,093130
	10	2230,41±47,69	0,001573	10	1951,06±17,24	0,308876
50	5	3482,16±58,77	0,003294	25	3320,67±23,18	2,911030
	10	4175,83±73,94	0,004123	25	3864,41±31,68	2,201643
	20	4798,46±59,70	0,007413	25	4713,37±32,32	2,447747
100	5	6789,97±70,93	0,008337	50	6260,08±25,37	21,796815
	10	7044,66±71,57	0,014096	50	6815,13±29,94	19,886336
	20	8606,34±116,25	0,027141	50	8021,79±30,59	23,970914
200	10	13388,23±117,94	0,062708	100	12831,82±44,80	248,030569
	20	15213,15±132,11	0,119019	100	14317,8±37,10	266,245554

Tabela 26 – Resultado com solução inicial gerada com PESO2 com *setup* em [1,10].

		Sol Inicial PALMER1		Sol Inicial PALMER1 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	365,96±4,25	0,000031	2	326,01±0,13	0,001461
10	3	712,34±10,01	0,000085	5	705,47±8,50	0,012495
	5	924,09±18,59	0,000109	5	944,84±18,26	0,015836
20	3	1535,71±10,16	0,000233	10	1433,76±16,76	0,090351
	5	1867,75±9,93	0,000326	10	1793,39±17,75	0,095471
	10	2447,61±23,45	0,000532	10	2352,62±25,35	0,103553
50	5	4144,04±8,27	0,001995	25	4075,03±27,70	1,932101
	10	4719,48±13,56	0,003630	25	4666,11±30,28	1,979271
	20	5586,98±15,03	0,006314	25	5687,55±35,17	2,080869
100	5	7918,49±11,01	0,006867	50	7544,35±36,83	16,956884
	10	8754,19±25,01	0,014682	50	8388,09±35,14	22,268311
	20	9944,86±16,47	0,025565	50	9811,40±37,27	24,001687
200	10	16218,70±13,56	0,051306	100	15866,32±50,62	222,720000
	20	17781,34±12,57	0,092810	100	17799,20±47,74	223,533673

Tabela 27 – Resultado com solução inicial gerada com PALMER1 com *setup* em [1,50].

		Sol Inicial PALMER2		Sol Inicial PALMER2 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	366,05±4,41	0,000036	2	326,01±0,16	0,001446
10	3	711,47±9,35	0,000086	5	705,38±9,10	0,012608
	5	925,47±18,75	0,000125	5	944,59±18,41	0,014568
20	3	1536,29±9,82	0,000268	10	1432,96±17,49	0,090416
	5	1866,71±9,89	0,000408	10	1792,28±17,77	0,094330
	10	2448,24±23,79	0,000887	10	2353,04±24,07	0,113030
50	5	4143,75±7,81	0,002415	25	4074,95±27,71	1,798863
	10	4718,29±14,35	0,004140	25	4664,49±31,22	1,975194
	20	5585,86±14,43	0,007595	25	5688,33±36,91	2,229621
100	5	7918,35±11,29	0,007800	50	7544,36±37,60	17,062345
	10	8753,61±25,05	0,013865	50	8388,01±36,82	17,123924
	20	9946,18±16,58	0,029723	50	9813,62±35,38	21,547921
200	10	16216,79±14,93	0,058764	100	15868,50±62,06	213,936812
	20	17779,77±11,79	0,102699	100	17789,33±46,82	221,467417

Tabela 28 – Resultado com solução inicial gerada com PALMER2 com *setup* em [1,50].

		Sol Inicial NEH1		Sol Inicial NEH1 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	373,83±7,79	0,000032	2	326,00±0,09	0,001694
10	3	787,38±25,70	0,000084	5	703,08±8,24	0,014593
	5	1048,06±28,54	0,000120	5	937,42±22,05	0,015216
20	3	1623,30±9,31	0,000242	10	1434,02±16,66	0,094416
	5	2022,44±21,30	0,000352	10	1791,67±17,83	0,105200
	10	2527,75±13,59	0,000568	10	2349,01±24,91	0,119809
50	5	4353,85±17,64	0,002035	25	4075,19±25,83	1,811590
	10	5194,12±30,12	0,003561	25	4664,51±30,97	2,174516
	20	5850,77±22,37	0,006659	25	5683,41±37,29	2,268233
100	5	8766,19±33,76	0,006528	50	7546,39±36,61	16,264190
	10	8918,05±11,75	0,012396	50	8386,16±36,59	17,308542
	20	10769,59±23,55	0,023467	50	9813,03±36,94	22,888147
200	10	17495,85±18,64	0,049296	100	15864,09±68,79	217,893347
	20	19240,02±17,79	0,090246	100	17790,02±45,58	219,064881

Tabela 29 – Resultado com solução inicial gerada com NEH1 com *setup* em [1,50].

		Sol Inicial NEH2		Sol Inicial NEH2 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	373,78±7,75	0,000035	2	326,00±0,00	0,001586
10	3	786,67±25,87	0,000091	5	702,97±8,93	0,012627
	5	1049,28±28,16	0,000117	5	936,60±22,67	0,013754
20	3	1623,17±10,06	0,000285	10	1433,19±16,62	0,096028
	5	2022,04±20,19	0,000403	10	1792,04±18,24	0,101050
	10	2528,43±14,30	0,000719	10	2350,23±25,23	0,119275
50	5	4353,45±17,97	0,002210	25	4073,85±27,71	1,754734
	10	5194,20±29,74	0,004387	25	4668,34±30,38	2,101510
	20	5850,96±22,23	0,008683	25	5680,31±38,28	2,607362
100	5	8766,95±33,91	0,007942	50	7548,19±34,91	17,263476
	10	8918,35±12,00	0,015306	50	8385,88±36,63	18,734233
	20	10770,93±24,92	0,029988	50	9812,07±37,05	21,677429
200	10	17498,57±17,19	0,069204	100	15872,62±49,84	253,433998
	20	19238,10±15,74	0,127141	100	17789,59±48,09	265,503802

Tabela 30 – Resultado com solução inicial gerada com NEH2 com *setup* em [1,50].

		Sol Inicial PESO1		Sol Inicial PESO1 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	416,15±43,42	0,000033	2	326,22±0,78	0,001399
10	3	860,34±52,16	0,000085	5	702,95±9,09	0,012458
	5	1144,60±30,27	0,000104	5	935,70±22,21	0,013319
20	3	1704,00±39,28	0,000218	10	1432,92±16,86	0,079571
	5	2118,72±40,40	0,000301	10	1791,85±18,27	0,085757
	10	2613,24±52,07	0,000637	10	2348,85±25,20	0,120828
50	5	4462,05±35,29	0,002106	25	4075,19±27,54	1,783184
	10	5343,48±46,95	0,003238	25	4665,49±31,46	1,862762
	20	5982,68±46,64	0,005926	25	5680,76±38,15	2,207933
100	5	8948,18±51,49	0,007458	50	7545,66±35,81	17,525780
	10	9012,41±39,93	0,014722	50	8385,88±37,99	19,874881
	20	9169,26±46,61	0,027493	50	8233,61±37,33	25,647164
200	10	17642,40±43,45	0,049419	100	15878,31±52,52	205,203428
	20	19413,41±57,15	0,093485	100	17789,71±43,65	225,741051

Tabela 31 – Resultado com solução inicial gerada com PESO1 com *setup* em [1,50].

		Sol Inicial PESO2		Sol Inicial PESO2 + AFSP		
Tar	Maq	*Makespan*	Tempo (seg)	Formigas	*Makespan*	Tempo (seg)
5	3	404,87±44,75	0,000034	2	326,30±0,90	0,001481
10	3	916,80±40,16	0,000088	5	702,75±9,49	0,012554
	5	1169,39±50,17	0,000116	5	938,58±21,69	0,014056
20	3	1773,09±56,33	0,000281	10	1432,71±17,39	0,098006
	5	2086,31±66,52	0,000359	10	1791,78±18,06	0,109682
	10	2598,55±59,99	0,000556	10	2349,25±24,31	0,115897
50	5	4512,90±91,41	0,002138	25	4073,42±27,82	1,763347
	10	5188,76±101,55	0,003855	25	4665,45±31,53	1,934638
	20	6049,33±119,27	0,006358	25	5683,99±37,48	1,968843
100	5	8761,06±158,86	0,008244	50	7545,44±37,61	18,337652
	10	9239,34±141,53	0,015037	50	8388,94±35,92	20,966198
	20	10737,38±169,66	0,027414	50	9812,92±36,68	21,481322
200	10	17695,34±181,45	0,059945	100	15873,94±55,65	249,653412
	20	19424,19±225,11	0,114037	100	17787,59±54,43	250,561143

Tabela 32 – Resultado com solução inicial gerada com PESO2 com *setup* em [1,50].